遷移式學習

Transfer learning

‘ 想看得更遠，就要站在巨人的身上 ’

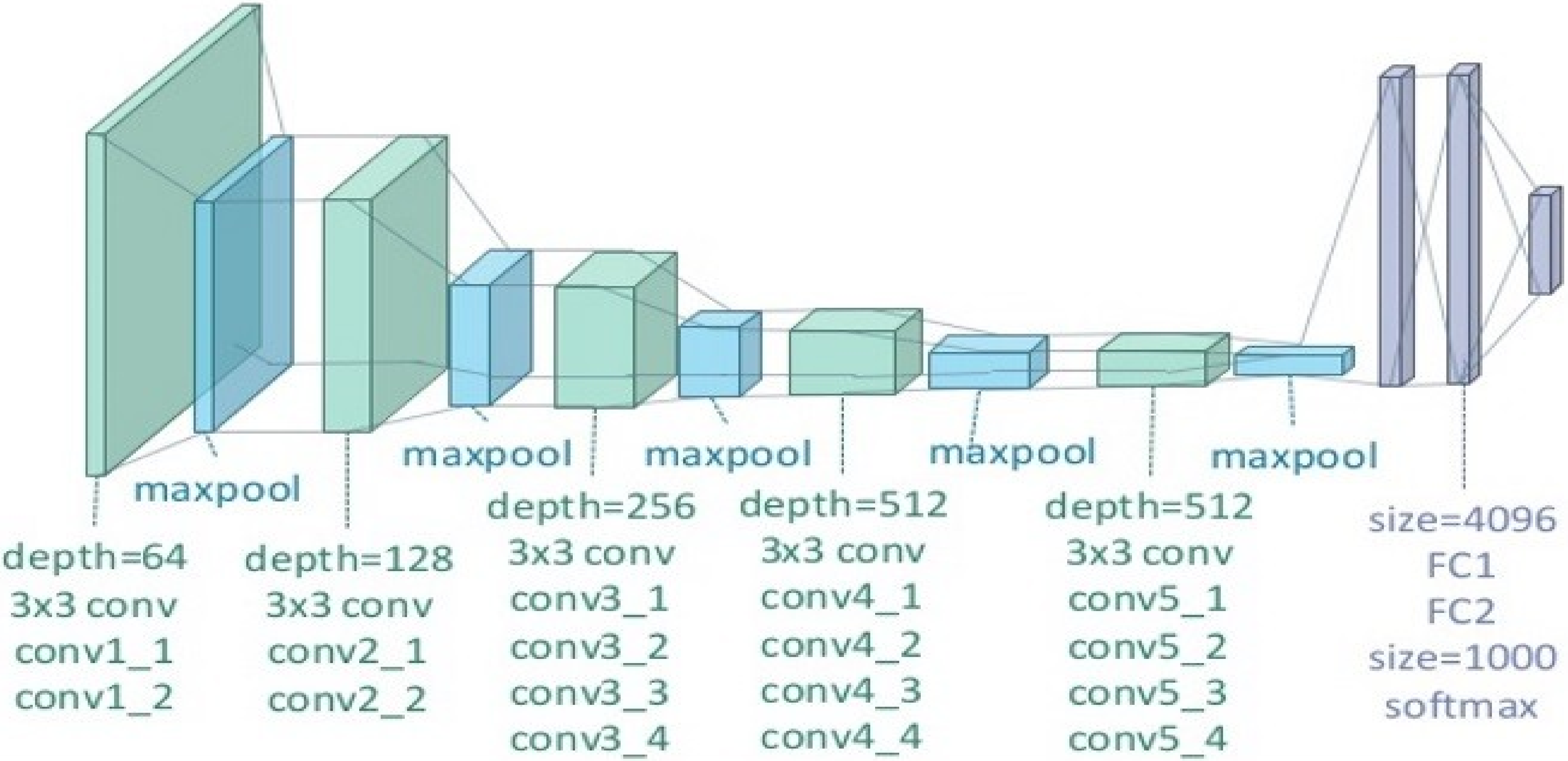
目錄
CONTENTS

01

深度學習基礎

Deep Learning Basics

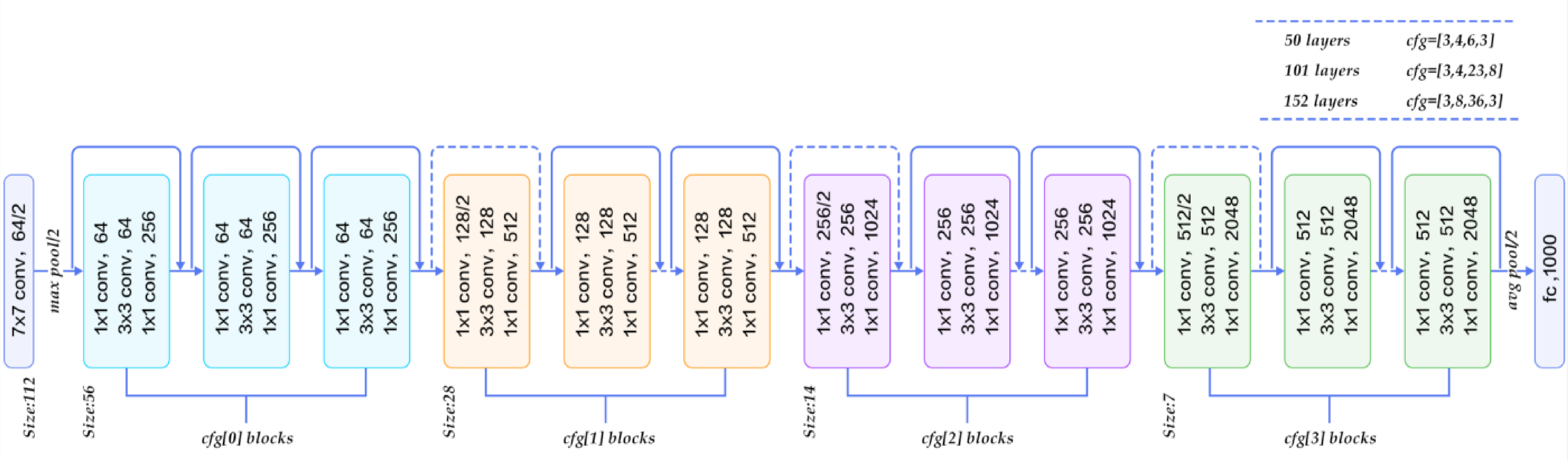
VGG 19


maxpool
maxpool
maxpool
maxpool
maxpool
depth=64
3x3 conv
conv1_1
conv1_2
depth=128
3x3 conv
conv2_1
conv2_2
depth=256
3x3 conv
conv3_1
conv3_2
conv3_3
conv3_4
depth=512
3x3 conv
conv4_1
conv4_2
conv4_3
conv4_4
depth=512
3x3 conv
conv5_1
conv5_2
conv5_3
conv5_4
size=4096
FC1
FC2
size=1000
softmax

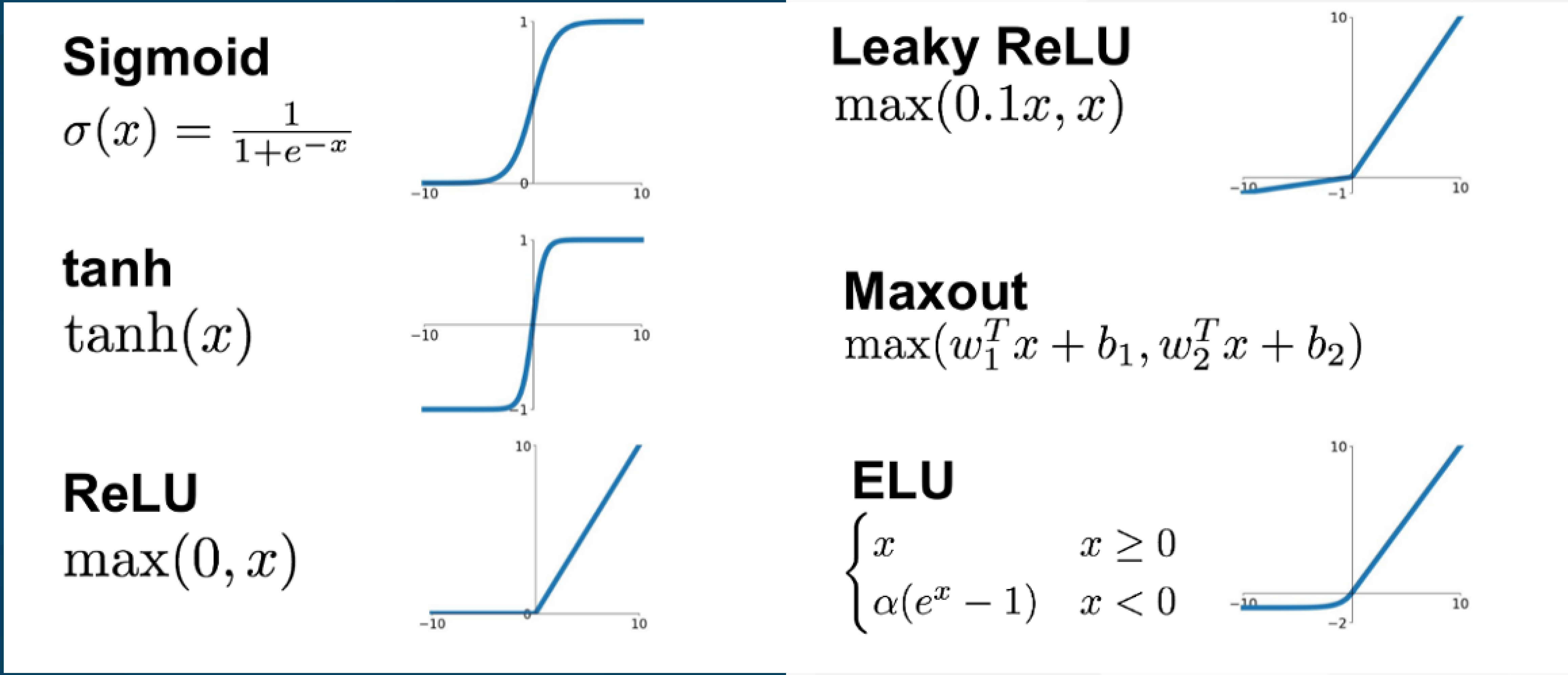
ImageNet 競賽的冠軍們

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.715	0.901	138,357,544	23
VGG19	549 MB	0.727	0.910	143,667,240	26
ResNet50	99 MB	0.759	0.929	25,636,712	168
InceptionV3	92 MB	0.788	0.944	23,851,784	159
InceptionResNetV2	215 MB	0.804	0.953	55,873,736	572
MobileNet	17 MB	0.665	0.871	4,253,864	88

ResNet


50 layers cfg=[3,4,6,3]
101 layers cfg=[3,4,23,8]
152 layers cfg=[3,8,36,3]
7x7 conv，64/2
max pool/2
1x1 conv，64
3x3 conv，64
1x1 conv，256
1x1 conv，128/2
3x3 conv，128
1x1 conv，512
1x1 conv，128
3x3 conv，128
1x1 conv，512
1x1 conv，256/2
3x3 conv，256
1x1 conv，1024
1x1 conv，256
3x3 conv，256
1x1 conv，1024
1x1 conv，512/2
3x3 conv，512
1x1 conv，2048
1x1 conv，512
3x3 conv，512
1x1 conv，2048
avg pool/2
fc ,1000
Size:112
Size:56
Size:28
Size:14
Size:7
cfg[0] blocks
cfg[1] blocks
cfg[2] blocks
cfg[3] blocks

Activation Function



Overview:CNN

CNN優點：

1） 輸入圖像和網路的拓撲結構能很好的吻合

2） 儘管使用較少參數，仍然有出色性能

3） 避免了顯式的特徵抽取，而隱式地從訓練數據中進行學習

4） 特徵提取和模式分類同時進行，並同時在訓練中產生，網路可以並行學習

5） 權值共享減少網路的訓練參數，降低了網路結構的複雜性，適用性更強

6） 無需手動選取特徵，訓練好權重，即得特徵，分類效果好

7） 可以直接輸入網路，避免了特徵提取和分類過程中，數據重建的複雜度

- 改良方法:
  1. Sparse interaction
  2. 參數共享
  3. 丟失率避免過度擬合
- 解決梯度消失
  1. Random Initialization 破壞weighting對稱性
  2. Bachnormalization
  3. Residual network

Overview:CNN

- CNN 特色 :

  Question 1: 如何有效地在圖像中找到圖案

  Question 2:如何能得到更多細節的特徵，如平移, 旋轉, 放大縮小 etc.

- 延伸應用:

  1D-CNN（心電圖、語音、股票……、電氣特性 I、v）

  3D-CNN（3D物體識別）

Part 1

01 深度學習三大框架 Keras/TF/Pytorch

02 手寫辨認 MNIST using Keras ANN

03 精品辨認 MNIST using Keras CNN

04 TF.keras 手寫辨認

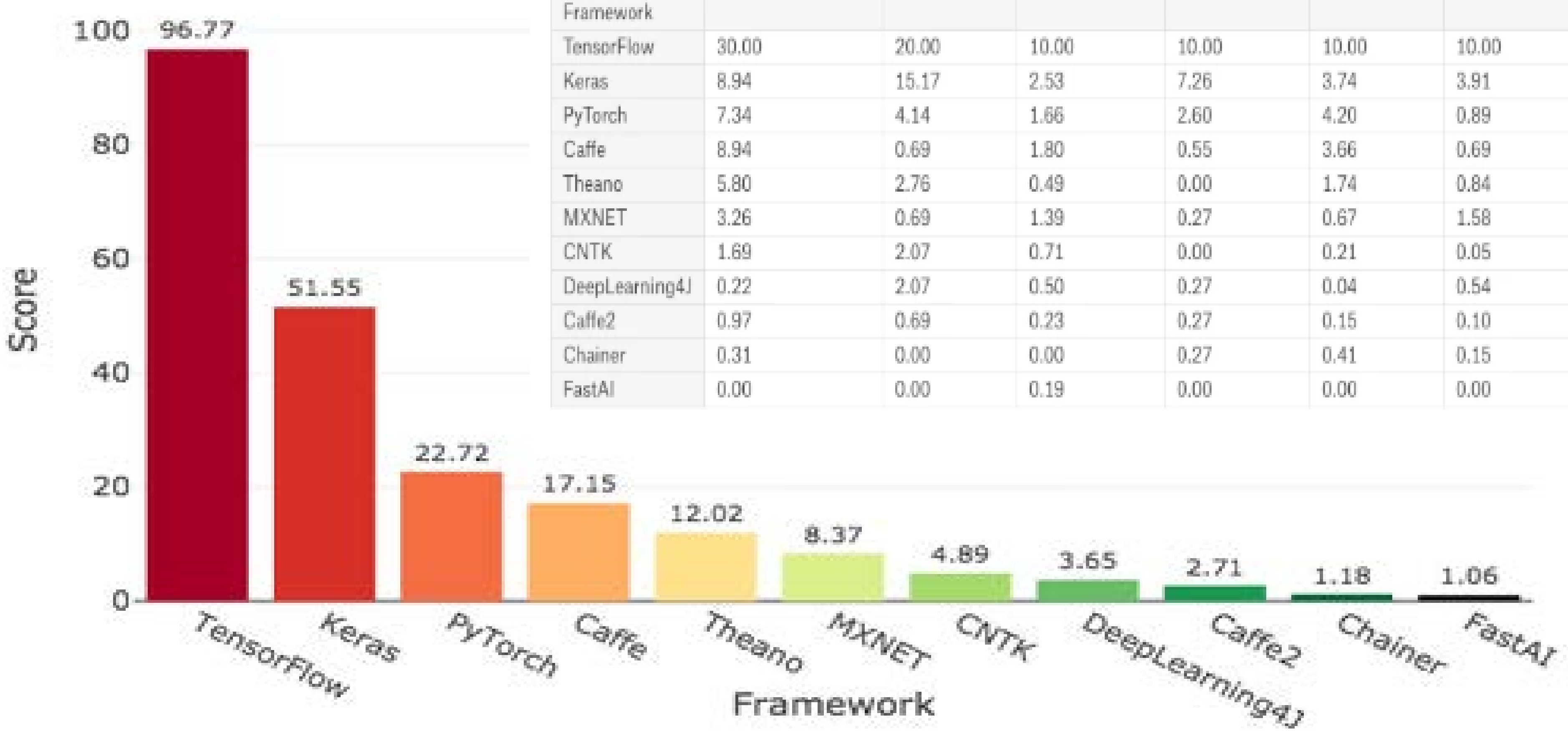
Part 1/ 01

DL 框架

2018 DL Framework

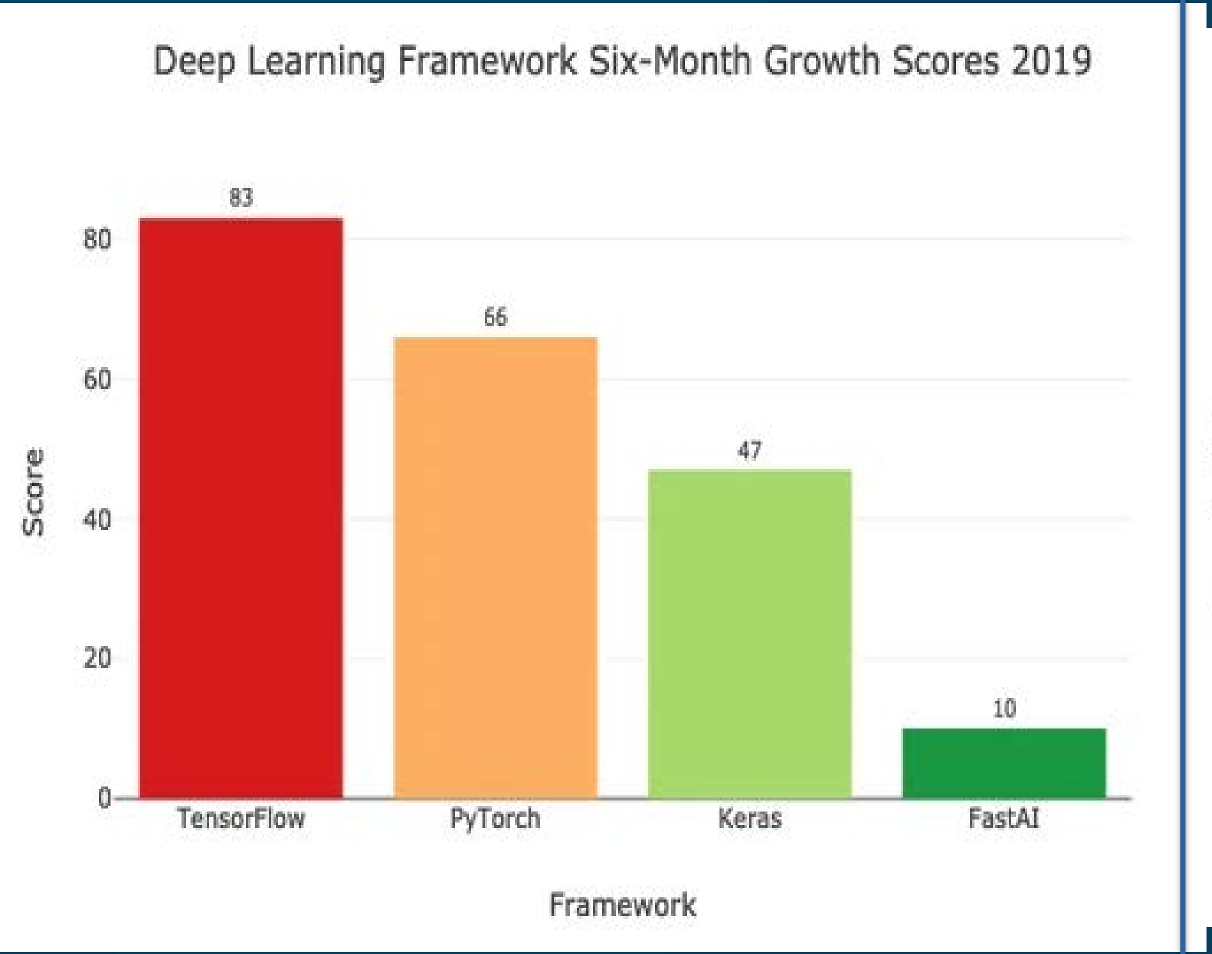
Deep Learning Framework Power Scores 2018

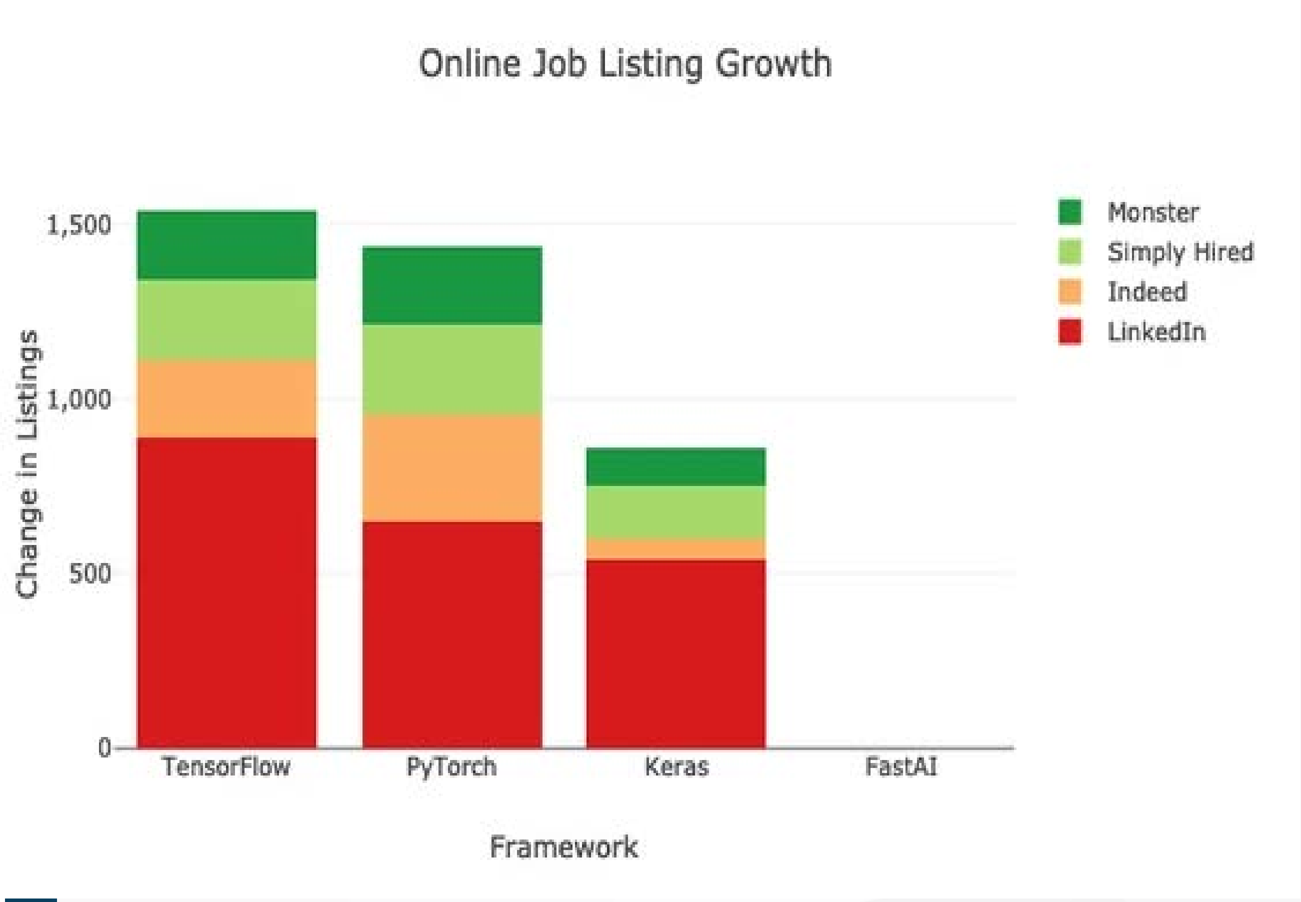
	Job Search Listings	Usage Survey	GitHub Activity	Search Volume	ArXiv Articles	Amazon Books	Medium Articles	Power Score
Framework								
TensorFlow	30.00	20.00	10.00	10.00	10.00	10.00	6.77	96.77
Keras	8.94	15.17	2.53	7.26	3.74	3.91	10.00	51.55
PyTorch	7.34	4.14	1.66	2.60	4.20	0.89	1.89	22.72
Caffe	8.94	0.69	1.80	0.55	3.66	0.69	0.82	17.15
Theano	5.80	2.76	0.49	0.00	1.74	0.84	0.40	12.02
MXNET	3.26	0.69	1.39	0.27	0.67	1.58	0.50	8.37
CNTK	1.69	2.07	0.71	0.00	0.21	0.05	0.17	4.89
DeepLearning4J	0.22	2.07	0.50	0.27	0.04	0.54	0.00	3.65
Caffe2	0.97	0.69	0.23	0.27	0.15	0.10	0.29	2.71
Chainer	0.31	0.00	0.00	0.27	0.41	0.15	0.02	1.18
FastAI	0.00	0.00	0.19	0.00	0.00	0.00	0.87	1.06



https://pttnews.cc/775cab551e

2019 DL Framework

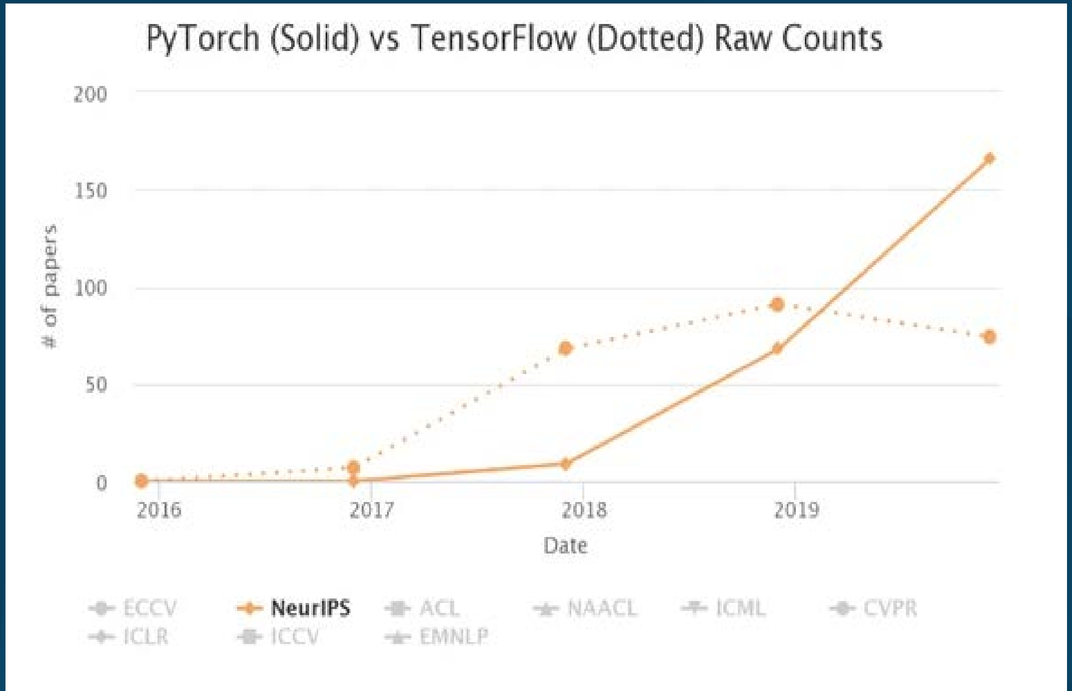




https://pttnews.cc/775cab551e

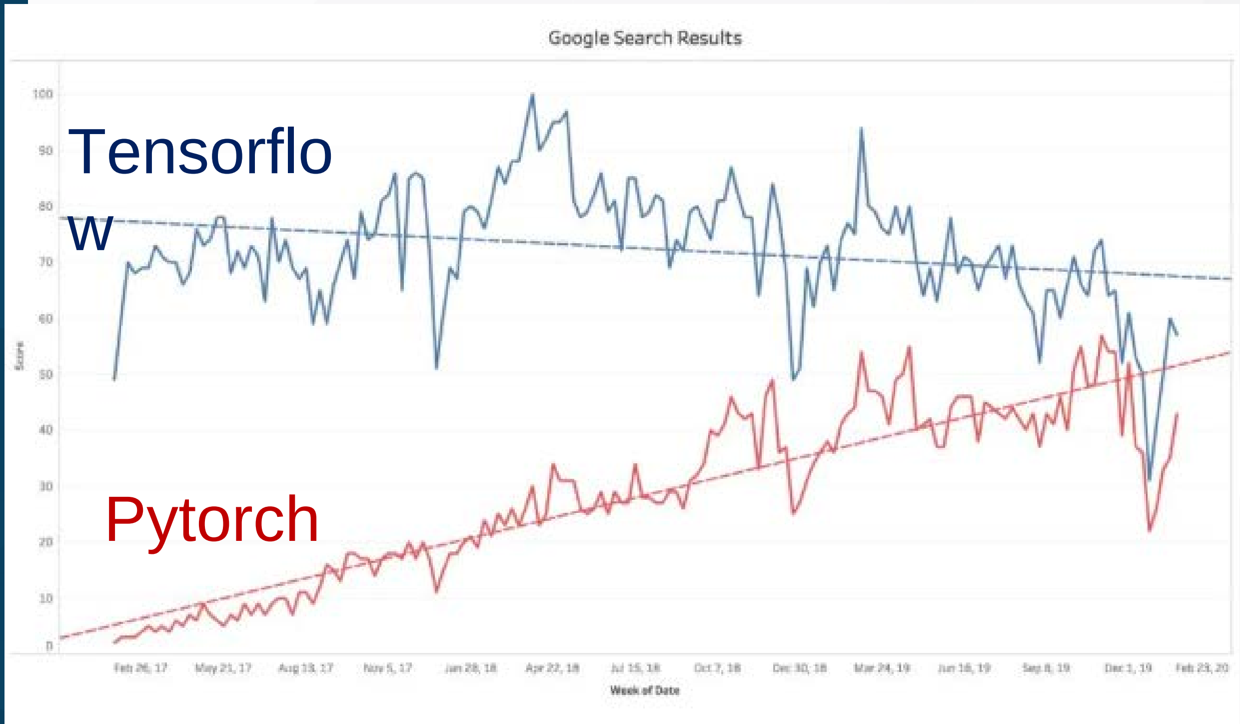
2020 研究論文數量

2019 NIPS



圖源：https://chillee.github.io/pytorch-vs-tensorflow/

Google Trend



https://pttnews.cc/775cab551e

三大框架比較

Keras
- 建模簡單直覺
- 輸入輸出方便

TF 1.0
- TF 1.0 計算圖

TF 2.0
- Eager Execution
- Keras 套件功能

Pytorch
- Pythonic 風格
- 動態計算圖
- 反向自動求導技術
- 研究社群Github

https://pttnews.cc/775cab551e / https://zhuanlan.zhihu.com/p/61926109

Part 2

01 深度學習三大框架 Keras/TF/Pytorch

02 手寫辨認 MNIST using Keras ANN

03 精品辨認 MNIST using Keras CNN

04 TF.keras 手寫辨認

Keras MNIST ANN

Keras

01 導入函式庫
載入資料

```
# 導入函式庫
import numpy as np
from keras.models import Sequential
from keras.datasets import mnist
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.utils import np_utils  # 用來後續將 label 標籤轉為 one-hot-encoding
from matplotlib import pyplot as plt

# 載入 MNIST 資料庫的訓練資料，並自動分為『訓練組』及『測試組』
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

https://pttnews.cc/775cab551e / https://zhuanlan.zhihu.com/p/61926109

Keras MNIST ANN

Keras

02 建立模型

```
# 建立簡單的線性執行的模型
model = Sequential()
# Add Input layer, 隱藏層(hidden layer) 有 256個輸出變數
model.add(Dense(units=256, input_dim=784, kernel_initializer='normal', activation='relu'))
# Add output layer
model.add(Dense(units=10, kernel_initializer='normal', activation='softmax'))

# 編譯: 選擇損失函數、優化方法及成效衡量方式
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

https://pttnews.cc/775cab551e / https://zhuanlan.zhihu.com/p/61926109

Keras MNIST ANN

Keras

02 建立模型

```
# 建立簡單的線性執行的模型
model = Sequential()
# Add Input layer, 隱藏層(hidden layer) 有 256個輸出變數
model.add(Dense(units=256, input_dim=784, kernel_initializer='normal', activation='relu'))
# Add output layer
model.add(Dense(units=10, kernel_initializer='normal', activation='softmax'))

# 編譯: 選擇損失函數、優化方法及成效衡量方式
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

https://pttnews.cc/775cab551e / https://zhuanlan.zhihu.com/p/61926109

Keras MNIST ANN

Keras

03 One-Hot encoding
資料降維
標準化
模型訓練

```
# 將 training 的 label 進行 one-hot encoding，例如數字 7 經過 One-hot encoding 轉換後是 0000001000，即第7個值為 1
y_TrainOneHot = np_utils.to_categorical(y_train)
y_TestOneHot = np_utils.to_categorical(y_test)

# 將 training 的 input 資料轉為2維
X_train_2D = X_train.reshape(60000, 28*28).astype('float32')
X_test_2D = X_test.reshape(10000, 28*28).astype('float32')

x_Train_norm = X_train_2D/255
x_Test_norm = X_test_2D/255

# 進行訓練, 訓練過程會存在 train_history 變數中
train_history = model.fit(x=x_Train_norm, y=y_TrainOneHot, validation_split=0.2, epochs=10, batch_size=800, verbose=2)
```

https://pttnews.cc/775cab551e / https://zhuanlan.zhihu.com/p/61926109

Keras MNIST ANN

Keras

04 顯示訓練結果
顯示預測結果

```
# 顯示訓練成果(分數)
scores = model.evaluate(x_Test_norm, y_TestOneHot)
print()
print("\t[Info] Accuracy of testing data = {:2.1f}%".format(scores[1]*100.0))

# 預測(prediction)
X = x_Test_norm[0:10,:]
predictions = np.argmax(model.predict(X), axis=-1)
# get prediction result
print(predictions)
```

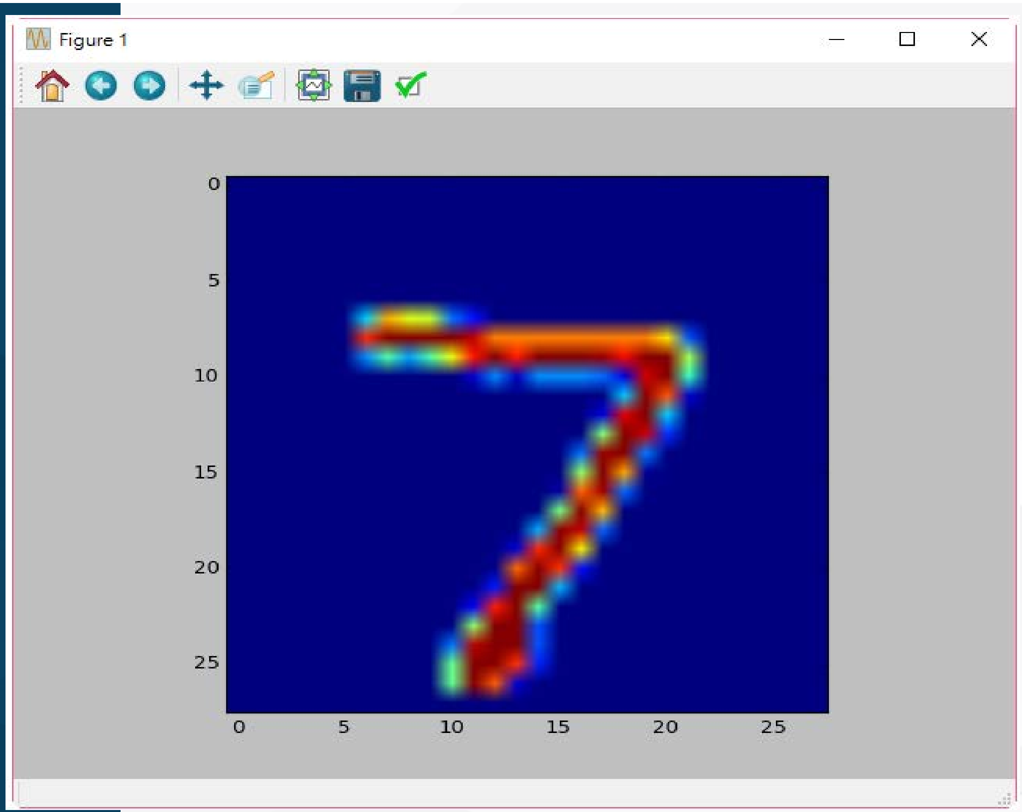
https://pttnews.cc/775cab551e / https://zhuanlan.zhihu.com/p/61926109

Keras MNIST ANN

Keras

05 顯示圖片

```
# 顯示 第一筆訓練資料的圖形，確認是否正確
plt.imshow(X_test[0])
plt.show()
```



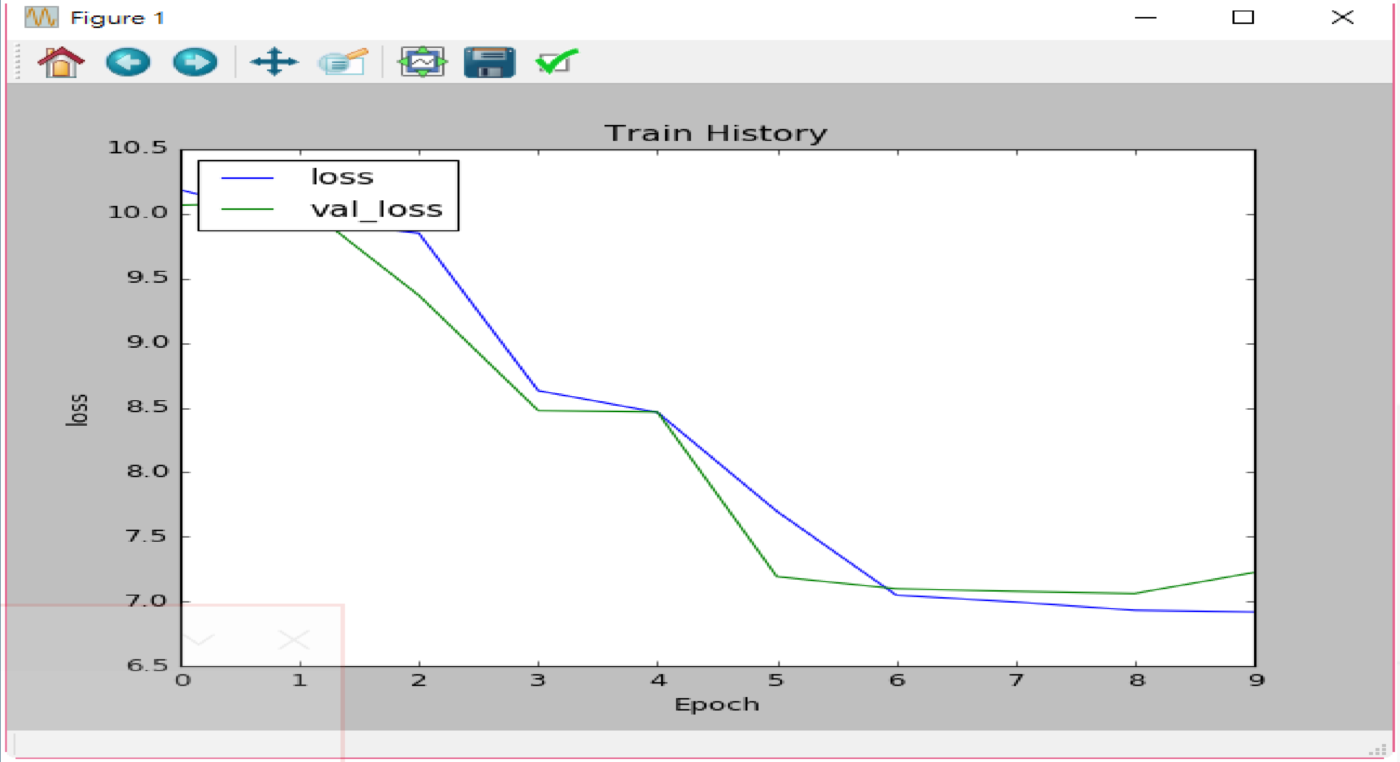
https://pttnews.cc/775cab551e / https://zhuanlan.zhihu.com/p/61926109

Keras MNIST ANN

Keras

06 收斂結果

```
plt.plot(train_history.history['loss'])
plt.plot(train_history.history['val_loss'])
plt.title('Train History')
plt.ylabel('loss')
plt.xlabel('Epoch')
plt.legend(['loss', 'val_loss'], loc='upper left')
plt.show()
```



https://pttnews.cc/775cab551e / https://zhuanlan.zhihu.com/p/61926109

Part 3

Keras MNIST CNN

Keras

1 導入函式庫
定義參數
載入資料

```
from __future__ import print_function
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K

# 定義梯度下降批量
batch_size = 128
# 定義分類數量
num_classes = 10
# 定義訓練週期
epochs = 12

# 定義圖像寬、高
img_rows, img_cols = 28, 28

# 載入 MNIST 訓練資料
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

https://pttnews.cc/775cab551e / https://zhuanlan.zhihu.com/p/61926109

Keras MNIST CNN

Keras

2 標準化
One-Hot encoding

```
# 轉換色彩 0~255 資料為 0~1
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255

# y 值轉成 one-hot encoding
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

https://pttnews.cc/775cab551e / https://zhuanlan.zhihu.com/p/61926109

Keras MNIST CNN

Keras

2 建立模型

```
# 建立簡單的線性執行的模型
model = Sequential()
# 建立卷積層，filter=32,即 output space 的深度, Kernal Size: 3x3, activation function 採用 relu
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
# 建立卷積層，filter=64,即 output size, Kernal Size: 3x3, activation function 採用 relu
model.add(Conv2D(64, (3, 3), activation='relu'))
# 建立池化層，池化大小=2x2，取最大值
model.add(MaxPooling2D(pool_size=(2, 2)))
# Dropout層隨機斷開輸入神經元，用於防止過度擬合，斷開比例:0.25
model.add(Dropout(0.25))
# Flatten層把多維的輸入一維化，常用在從卷積層到全連接層的過渡。
model.add(Flatten())
# 全連接層: 128個output
model.add(Dense(128, activation='relu'))
# Dropout層隨機斷開輸入神經元，用於防止過度擬合，斷開比例:0.5
model.add(Dropout(0.5))
# 使用 softmax activation function，將結果分類
model.add(Dense(num_classes, activation='softmax'))

# 編譯: 選擇損失函數、優化方法及成效衡量方式
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])
```

https://pttnews.cc/775cab551e / https://zhuanlan.zhihu.cor

Keras MNIST CNN

Keras

03 模型訓練

```
# 進行訓練, 訓練過程會存在 train_history 變數中
train_history = model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
```

https://pttnews.cc/775cab551e / https://zhuanlan.zhihu.com/p/61926109

Keras MNIST CNN

Keras

04 顯示訓練結果

```
# 顯示損失函數、訓練成果(分數)
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
).9863
Epoch 5/12
60000/60000 [==============================] - 236s 4ms/step - loss: 0.0607 - acc: 0.9818 - val_loss: 0.0351 - val_acc:
).9888
Epoch 6/12
60000/60000 [==============================] - 239s 4ms/step - loss: 0.0562 - acc: 0.9831 - val_loss: 0.0336 - val_acc:
).9892
Epoch 7/12
60000/60000 [==============================] - 239s 4ms/step - loss: 0.0487 - acc: 0.9854 - val_loss: 0.0349 - val_acc:
).9882
Epoch 8/12
60000/60000 [==============================] - 240s 4ms/step - loss: 0.0452 - acc: 0.9863 - val_loss: 0.0319 - val_acc:
).9897
Epoch 9/12
60000/60000 [==============================] - 244s 4ms/step - loss: 0.0437 - acc: 0.9864 - val_loss: 0.0305 - val_acc:
).9897
Epoch 10/12
60000/60000 [==============================] - 240s 4ms/step - loss: 0.0403 - acc: 0.9882 - val_loss: 0.0284 - val_acc:
).9903
Epoch 11/12
60000/60000 [==============================] - 236s 4ms/step - loss: 0.0379 - acc: 0.9888 - val_loss: 0.0299 - val_acc:
).9904
Epoch 12/12
60000/60000 [==============================] - 251s 4ms/step - loss: 0.0392 - acc: 0.9881 - val_loss: 0.0278 - val_acc:
).9911
Test loss: 0.027754753280
Test accuracy: 0.9911

D:\0_DataMining\0_ITHome>_
微軟注音 半 :
```

https://pttnews.cc/775cab551e / https://zhuanlan.zhihu.co

Keras MNIST CNN

Keras

05 模型存檔

```
# 模型結構存檔
from keras.models import model_from_json
json_string = model.to_json()
with open("cnn.config", "w") as text_file:
    text_file.write(json_string)


# 模型訓練結果存檔
model.save_weights("cnn.weight")
```

https://pttnews.cc/775cab551e / https://zhuanlan.zhihu.com/p/61926109

Keras MNIST CNN

Keras

06 計算混淆矩陣

```
# 計算『混淆矩陣』(Confusion Matrix)，顯示測試集分類的正確及錯誤總和數
import pandas as pd
predictions = model.predict_classes(x_test)
pd.crosstab(y_test_org, predictions, rownames=['實際值'], colnames=['預測值'])
```

預測值 實際值	0	1	2	3	4	5	6	7	8	9
0	976	0	2	0	0	0	0	1	1	0
1	0	1131	2	1	0	0	0	0	1	0
2	1	0	1029	0	0	0	0	2	0	0
3	0	0	3	1004	0	1	0	0	1	1
4	0	0	1	0	970	0	4	0	1	6
5	2	0	0	8	0	879	3	0	0	0
6	7	2	0	0	1	2	946	0	0	0
7	0	2	7	1	0	0	0	1015	1	2
8	5	0	2	1	0	0	0	1	963	2
9	1	1	0	2	2	3	0	1	3	996

https://pttnews.cc/775cab551e / https://zhuanlan.zhihu.com/p/61926109

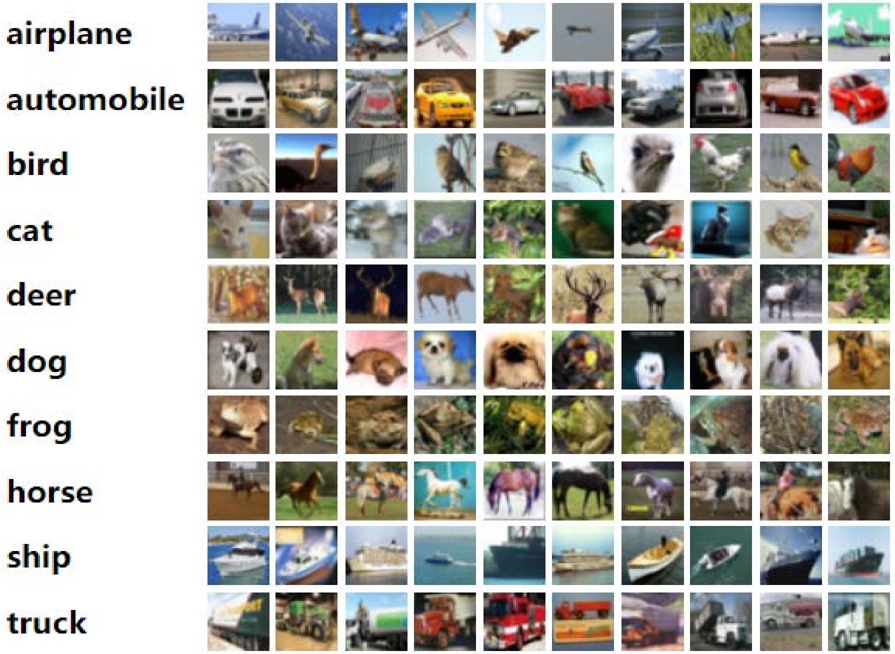
TF.Keras MNIST CNN

TF.Keras

```python
import tensorflow as tf
mnist = tf.keras.datasets.mnist

# 匯入 MNIST 手寫阿拉伯數字 訓練資料
(x_train, y_train),(x_test, y_test) = mnist.load_data()
# 特徵縮放，使用常態化(Normalization)，公式 = (x - min) / (max - min)
# 顏色範圍：0~255，所以，公式簡化為 x / 255
x_train, x_test = x_train / 255.0, x_test / 255.0

# 建立模型
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

# 設定優化器(optimizer)、損失函數(loss)、效能衡量指標(metrics)的類別
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# 模型訓練
model.fit(x_train, y_train, epochs=5)
# 模型評估，打分數
model.evaluate(x_test, y_test)
```

https://pttnews.cc/775cab551e / https://zhuanlan.zhihu.com/p/

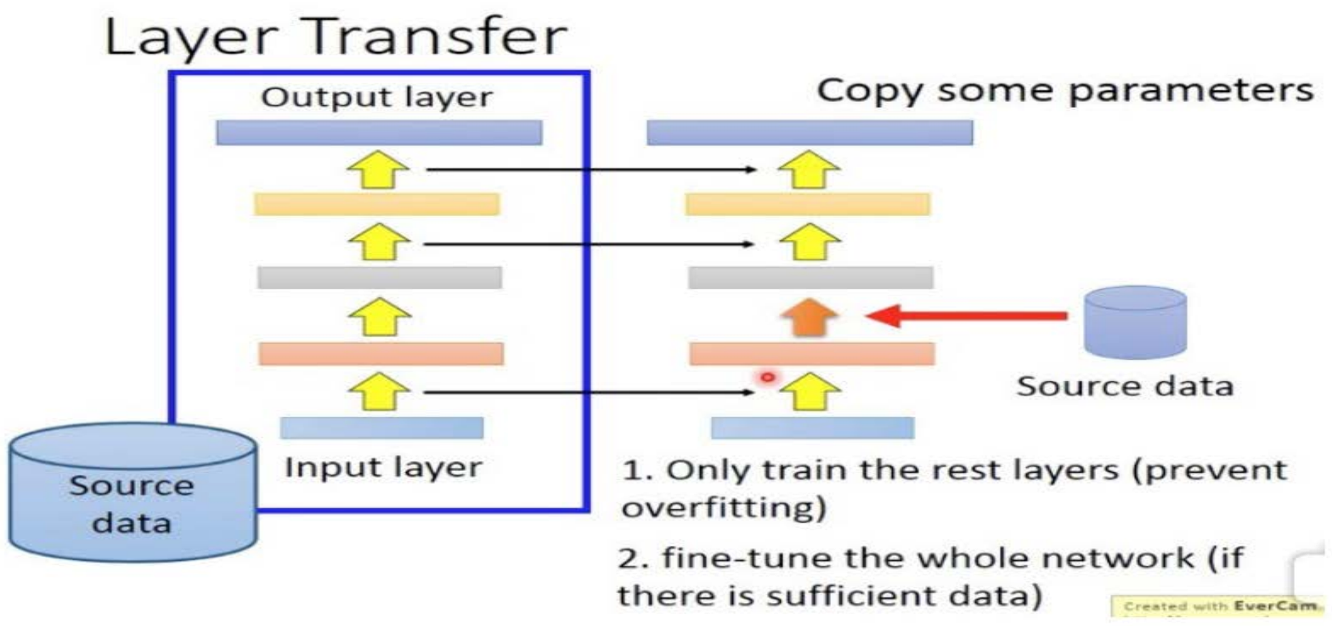
MNIST to CIFAR-10



CIFAR-10

Transfer Learning
(How to use pretrained model)

將source
去萃取其
有萃取的
去訓練學



02

遷移式學習環境設定

請配合Colab ipynb程式

1.

```
!pip3 install torch torchvision
!pip3 install gradio
import gradio as gr
from torchvision import datasets, transforms, models
import torch
!git clone https://github.com/chandrikadeb7/Face-Mask-Detection.git
```

- 載入pytorch & gradio套件
- 載入要mask_detection圖檔

2.

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

- 若cuda環境可用 則使用GPU計算 否則使用CPU

3.

```
transform_train = transforms.Compose([transforms.Resize((224,224)),
                                      transforms.RandomHorizontalFlip(),
                                      transforms.RandomAffine(0, shear=10, scale=(0.8,1.2)),
                                      transforms.ColorJitter(brightness=1, contrast=1, saturation=1),
                                      transforms.ToTensor(), # 轉為tensor數據
                                      transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)) # 資料正規化
                               ])

transform = transforms.Compose([transforms.Resize((224,224)),
                               transforms.ToTensor(),
                               transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
                               ])
```

- 針對訓練集做資料擴增
- 將資料轉成Tensor數據且正規化

4.

```
training_dataset = datasets.ImageFolder('Face-Mask-Detection/dataset/', transform=transform_train)
validation_dataset = datasets.ImageFolder('Face-Mask-Detection/dataset/', transform=transform)

training_loader = torch.utils.data.DataLoader(training_dataset, batch_size=20, shuffle=True)
validation_loader = torch.utils.data.DataLoader(validation_dataset, batch_size = 20, shuffle=False)
```

- 分為訓練集與測試集 並做transform
- 每20筆資料為一個batch

02

Transfer

轉移

1.

```
model = models.vgg16(pretrained=True)
```

➢載入模型與pytorch訓練好的權重

```
for param in model.features.parameters():
    param.requires_grad = False
```

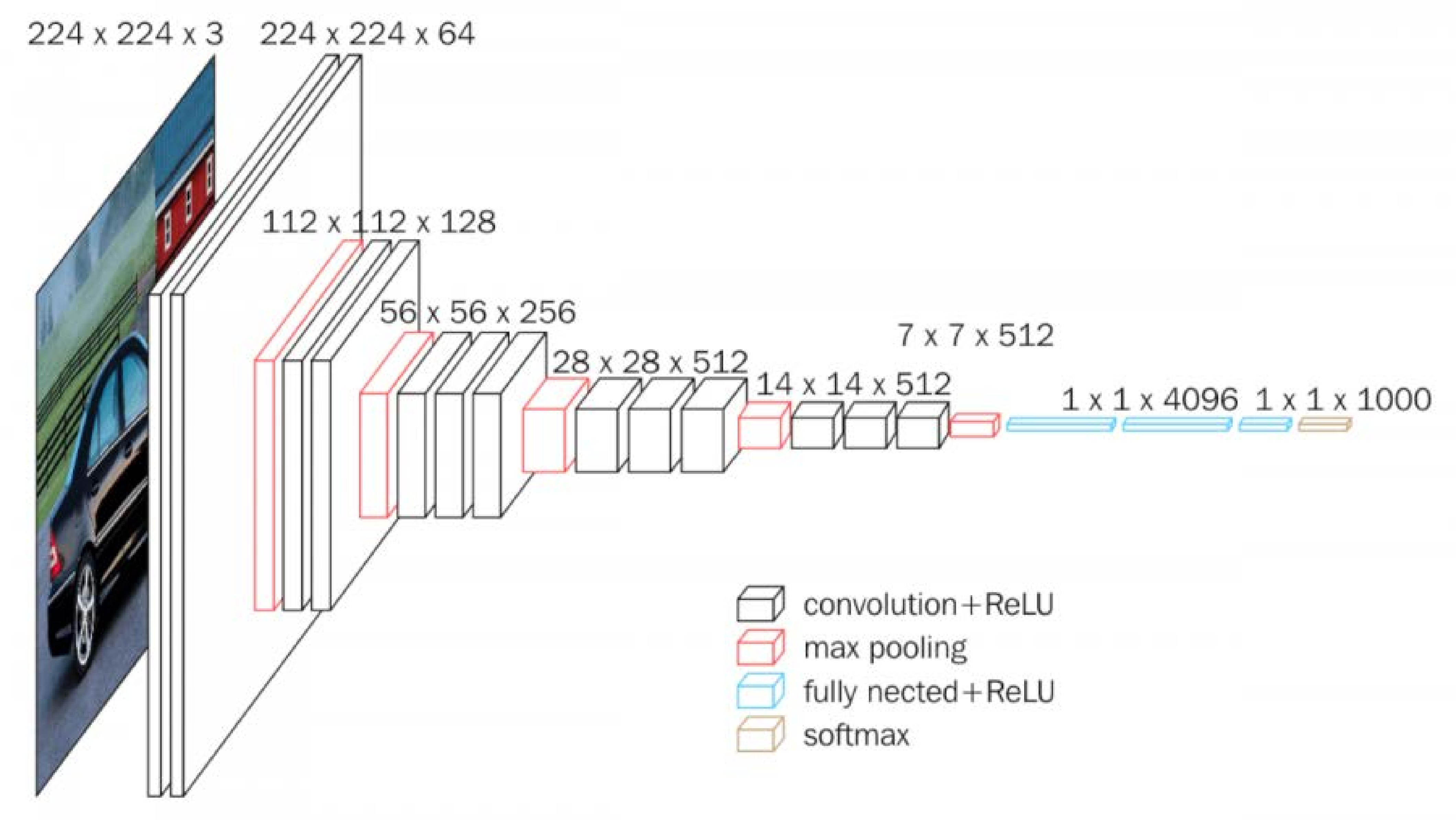
➢只訓練最後一層全連接層的權重

2.

```
import torch.nn as nn
classes=('mask', 'no_mask')
n_inputs = model.classifier[6].in_features
# 第六層(最後一層)原輸入node數
last_layer = nn.Linear(n_inputs, len(classes))
# layer的輸入:4096, 輸出:2
model.classifier[6] = last_layer
# 最後一層設為自定義的layer
model.to(device)
# 模型加載到指定設備
```

➢將原本VGG16的最後一層從1000個輸出改成2個

3.


224 x 224 x 3
224 x 224 x 64
112 x 112 x 128
56 x 56 x 256
28 x 28 x 512
14 x 14 x 512
7 x 7 x 512
1 x 1 x 4096
1 x 1 x 1000
convolution+ReLU
max pooling
fully nected+ReLU
softmax

4.

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
```

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=2, bias=True)
  )
)
```

03

Train

訓練

1.

```
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr = 0.0001)
```

- 損失函數用crossentropy
- 學習優化器用adam
- learning rate = 0.0001

2.

```python
epochs = 5
for e in range(epochs):
    running_loss = 0.0
    running_corrects = 0.0
    val_running_loss = 0.0
    val_running_corrects = 0.0

    for inputs,labels in training_loader:
        inputs = inputs.to(device)
        labels = labels.to(device)

        outputs = model(inputs) # 1
        loss = criterion(outputs, labels) # 2
        optimizer.zero_grad() # 3
        loss.backward() # 4
        optimizer.step() # 5
```

```python
_, preds = torch.max(outputs, 1)
# 輸出所在行最大值的欄位(預測機率最高)
running_loss += loss.item()
running_corrects += torch.sum(preds == labels.data)
```

- 1. 將data傳入model進行forward propagation
- 2. 計算loss
- 3. 清空前一次的gradient
- 4. 根據loss進行back propagation，計算gradient
- 5. 做gradient descent

- 計算loss值與accurncy

3.

```
else:
  with torch.no_grad():
    for val_inputs, val_labels in validation_loader:
      val_inputs = val_inputs.to(device)
      val_labels = val_labels.to(device)
      val_outputs = model(val_inputs)
      val_loss = criterion(val_outputs, val_labels)

      _, val_preds = torch.max(val_outputs, 1)
      val_running_loss += val_loss.item()
      val_running_corrects += torch.sum(val_preds == val_labels.data)
```

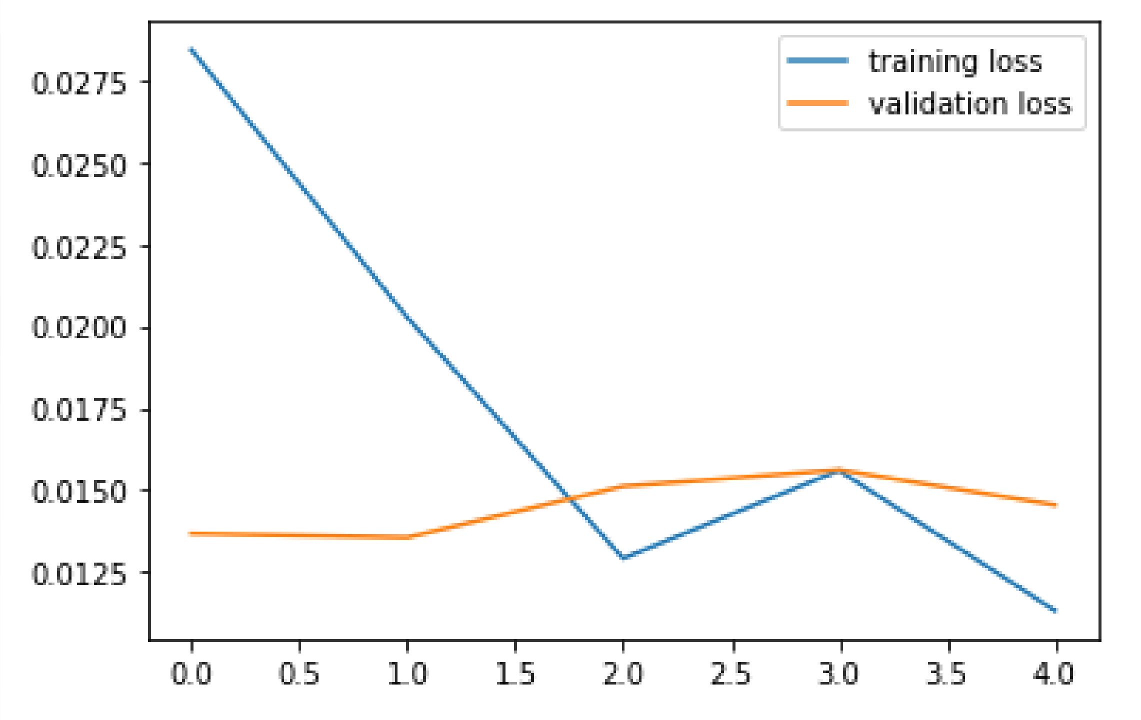
➢當不需要再計算梯度時用測試集測試模型

```
epoch_loss = running_loss/len(training_loader.dataset)
epoch_acc = running_corrects.float()/ len(training_loader.dataset)
running_loss_history.append(epoch_loss)
running_corrects_history.append(epoch_acc)

val_epoch_loss = val_running_loss/len(validation_loader.dataset)
val_epoch_acc = val_running_corrects.float()/ len(validation_loader.dataset)
val_running_loss_history.append(val_epoch_loss)
val_running_corrects_history.append(val_epoch_acc)
```

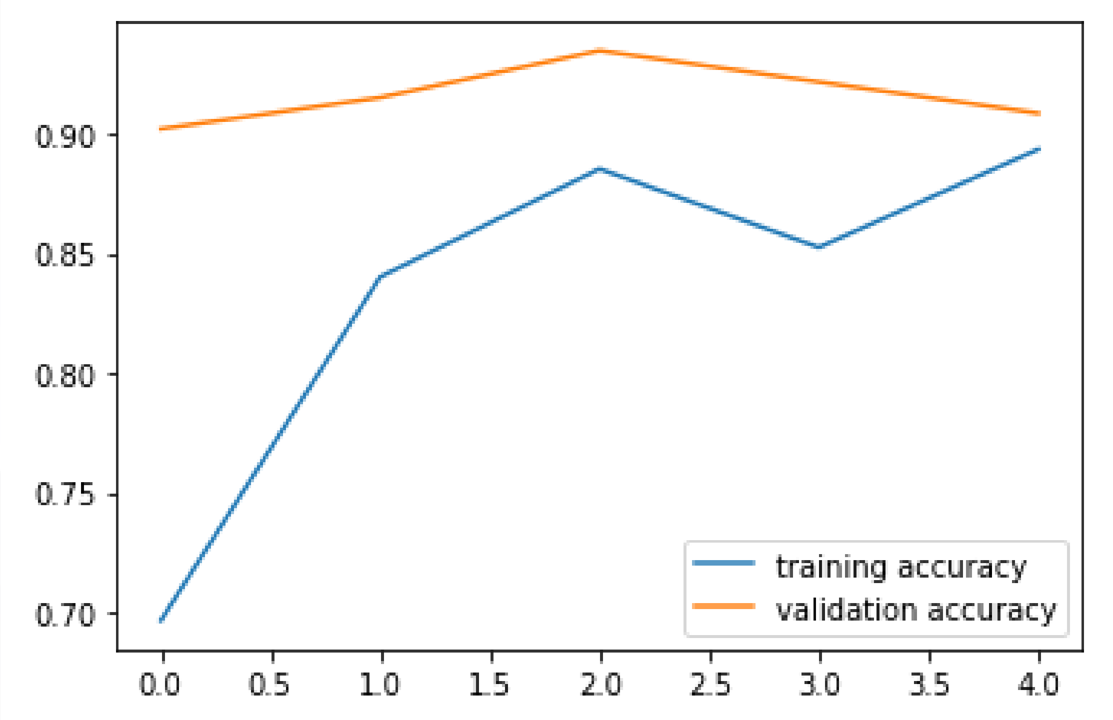
➢計算訓練集與測試集之

Loss與accurency

3.



training loss
validation loss
0.0275
0.0250
0.0225
0.0200
0.0175
0.0150
0.0125
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0



0.90
0.85
0.80
0.75
0.70
training accuracy
validation accuracy
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

04

Gradio

視覺化部屬

1.

```
def predict(img):
    labels = ['mask', 'no_mask']
    image = transforms.ToTensor()(img).unsqueeze(0)
    prediction = torch.nn.functional.softmax(model(image)[0], dim=0)
    confidences = {labels[i]: float(prediction[i]) for i in range(2)}
    return confidences
```

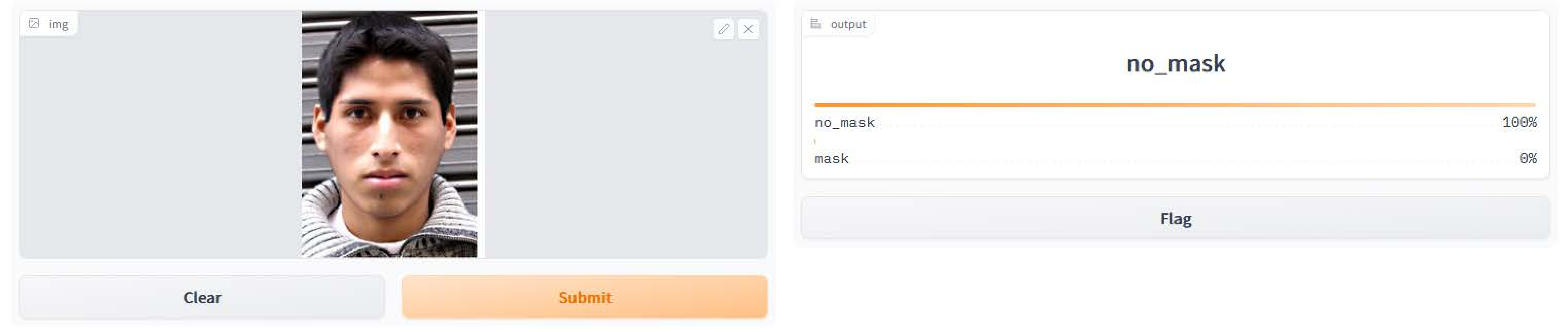
- **image轉為tensor數據**
- **每個維度進行softmax運算**

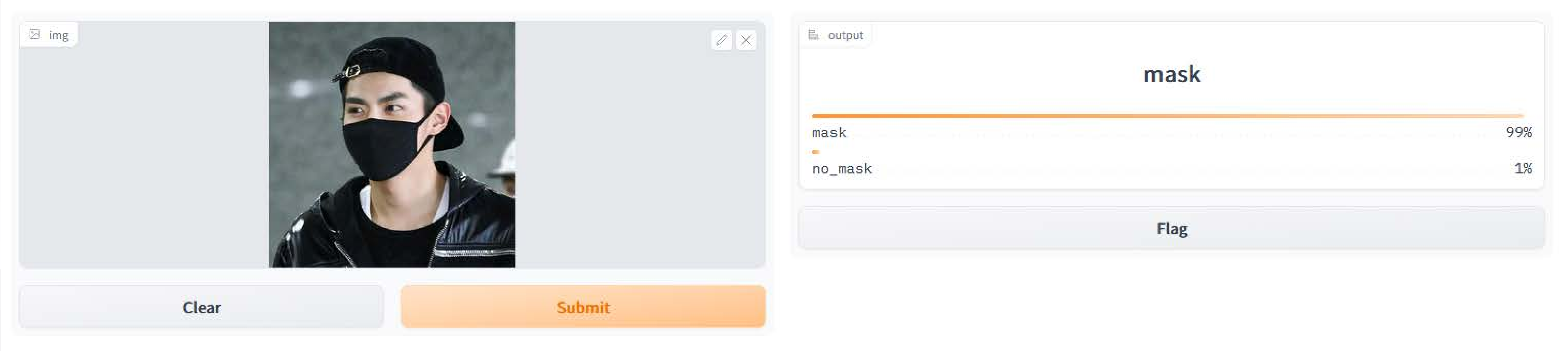
2.

```
gr.Interface(fn=predict,
             inputs=gr.Image(type="pil"),
             outputs=gr.Label(num_top_classes=2)).launch()
```

- 創建gradio接口
- 連接predict function

3.


img
output
no_mask
no_mask 100%
mask 0%
Flag
Clear
Submit


img
output
mask
mask 99%
no_mask 1%
Flag
Clear
Submit