機器學習於材料資訊的應用
Machine Learning on Material Informatics

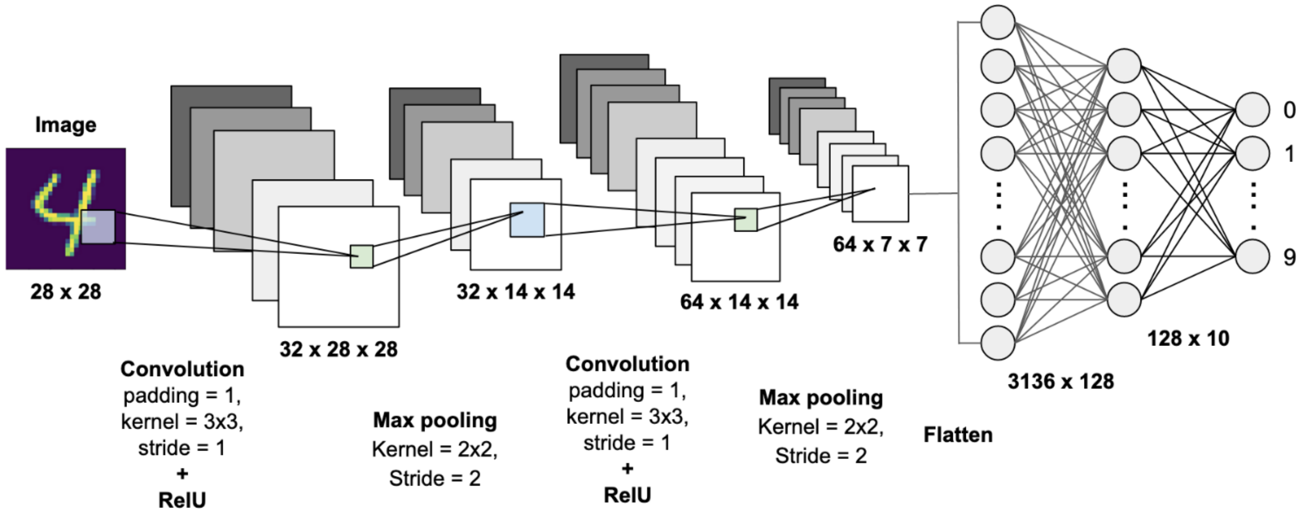
陳南佑(NAN-YOW CHEN)

nanyow@narlabs.org.tw

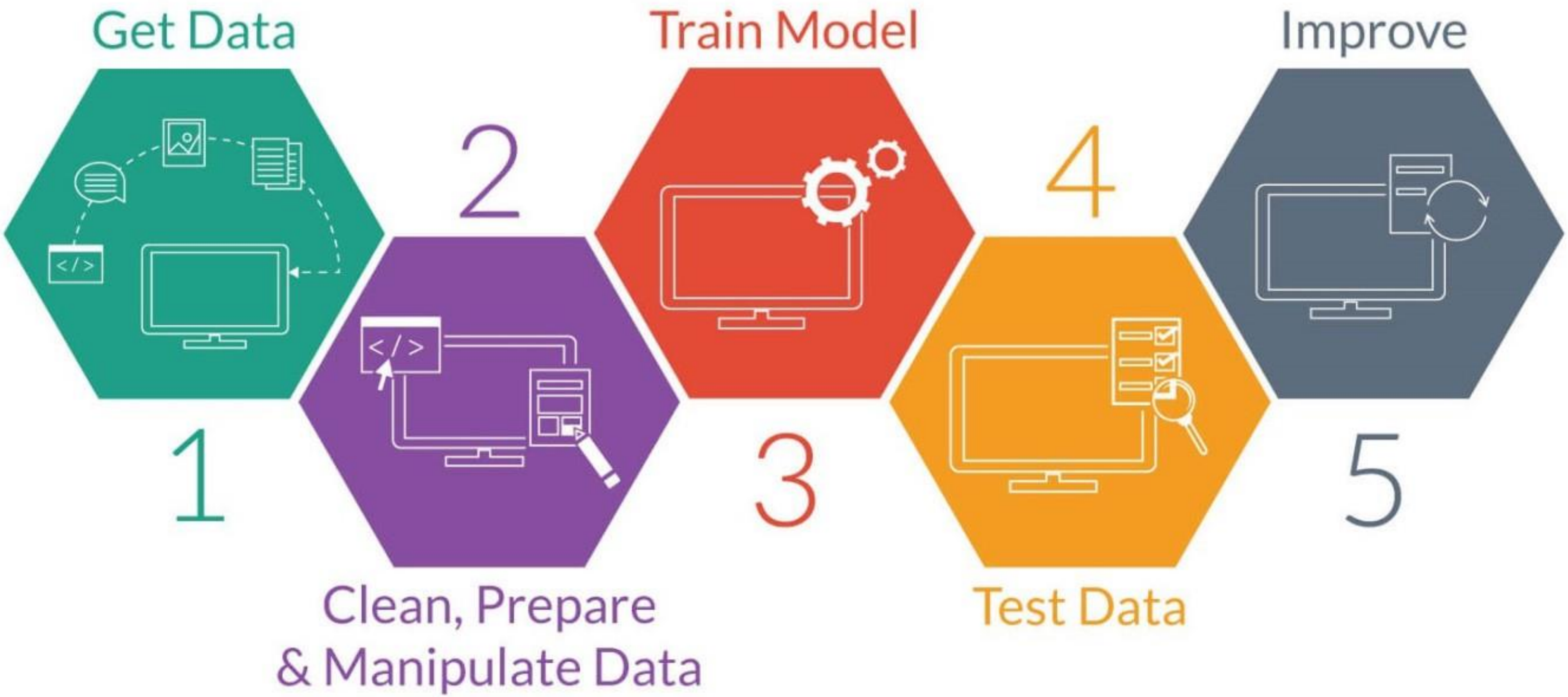
楊安正(AN-CHENG YANG)

acyang@narlabs.org.tw

Handwritten Digits classification by CNN



http://yann.lecun.com/exdb/mnist/


Get Data
1
2
Clean, Prepare
& Manipulate Data
Train Model
3
4
Test Data
Improve
5

Import module

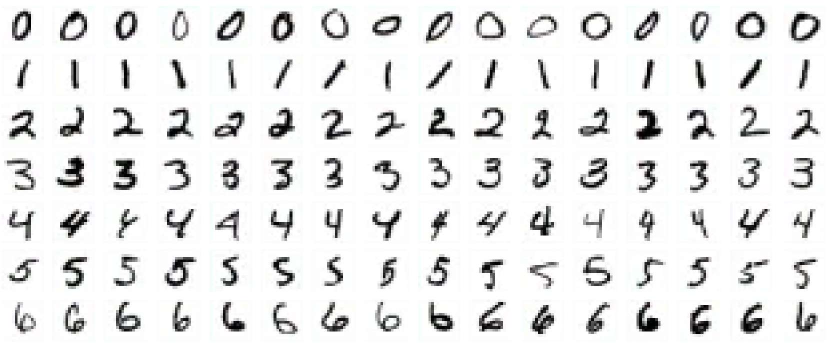
```python
#!pip install tensorflow-addons
import tensorflow as tf
import tensorflow_addons as tfa
print("TensorFlow version:", tf.__version__)

gpus = tf.config.list_physical_devices('GPU')
for gpu in gpus:
    print("Name:", gpu.name, "  Type:", gpu.device_type)

from tensorflow.keras.layers import Dense, Flatten, Conv2D
from tensorflow.keras import Model
```

TensorFlow SIG 附加元件是社群貢獻的存放區，符合完善的 API 模式，但實作了核心 TensorFlow 所沒有的新功能。

Get Data

THE MNIST DATABASE

of handwritten digits



MNIST database 由兩種資料來源組成NIST's Special Database 3(SD-3)和 Special Database 1(SD-1)。
SD-3 品質比SD-1更乾淨更容易分類。

http://yann.lecun.com/exdb/mnist/

1. 手動下載 wget curl
2. https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/tutorials/mnist/input_data.py (即將廢棄)
3. https://www.tensorflow.org/api_docs/python/tf/keras/datasets/mnist/load_data
4. …

Get Data

```python
# https://www.tensorflow.org/api_docs/python/tf/keras/datasets/mnist/load_data
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

# Add a channels dimension
# 三個點是切片操作，表示前面所有維度
# x_train[..., tf.newaxis] x_train[:, :, tf.newaxis]兩者等價
x_train = x_train[..., tf.newaxis].astype("float32")
x_test = x_test[..., tf.newaxis].astype("float32")
print(x_train.shape)
```

Tensorflow Dataset

- 正式的名稱為 tf.data API，它是一個 Python Generator，可以視需要逐批讀取必要資料，不必一股腦將資料全部讀取放在記憶體。
- 它還有快取(Cache)、預取(Prefetch)、篩選(Filter)、轉換(Map)等功能

Get Data

```
# https://www.tensorflow.org/guide/data_performance
# 一次取 10000 個洗牌，取完，再抽 10000 個洗牌
# batch(32)：一次取 32 個

train_ds = tf.data.Dataset.from_tensor_slices(
    (x_train, y_train)).shuffle(10000).batch(32)

test_ds = tf.data.Dataset.from_tensor_slices((x_test, y_test)).batch(32)
```

Model building-Model Subclassing

```
# Model Subclassing
class network(tf.keras.Model):
    def __init__(self):
        super(network,self).__init__()
        …
    def call(self,x):
        …
        return predict
model = network()
```

Train Model

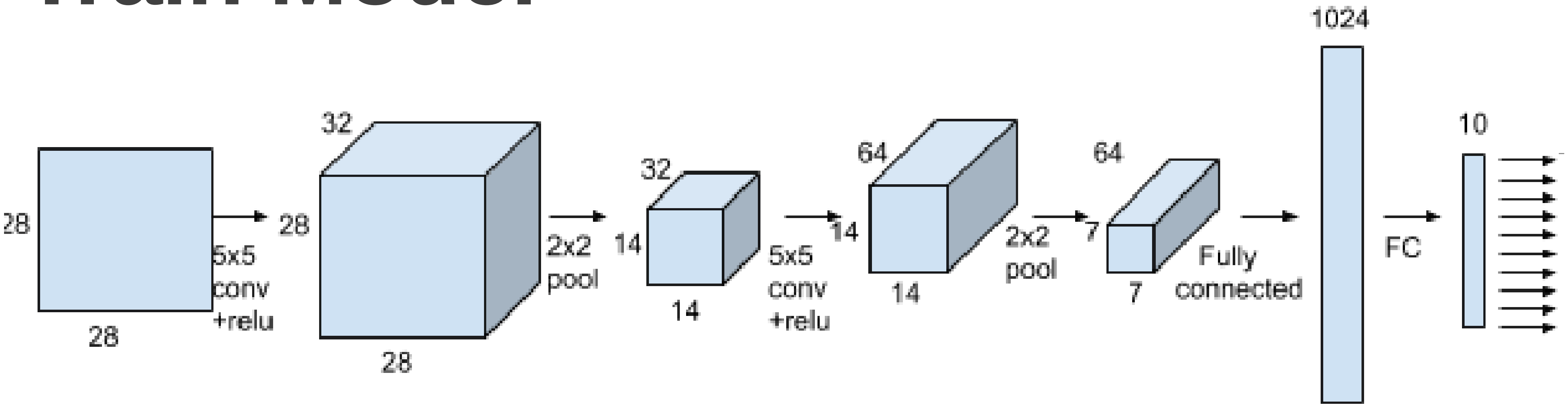


Figure D.2: Network architecture for MNIST classifier CNN

```
def __init__(self):
    super(MyModel, self).__init__()
    self.conv1 = Conv2D(filters=32, kernel_size=(5, 5), padding='same', activation='relu')
    self.maxpool1 = MaxPooling2D(pool_size=(2, 2), strides=2)
    self.conv2 = Conv2D(filters=64, kernel_size=(5, 5), padding='same', activation='relu')
    self.maxpool2 = MaxPooling2D(pool_size=(2, 2), strides=2)
    self.flatten = Flatten()
    self.d1 = Dense(1024, activation='relu')
    self.d2 = Dense(10)
```

Train Model

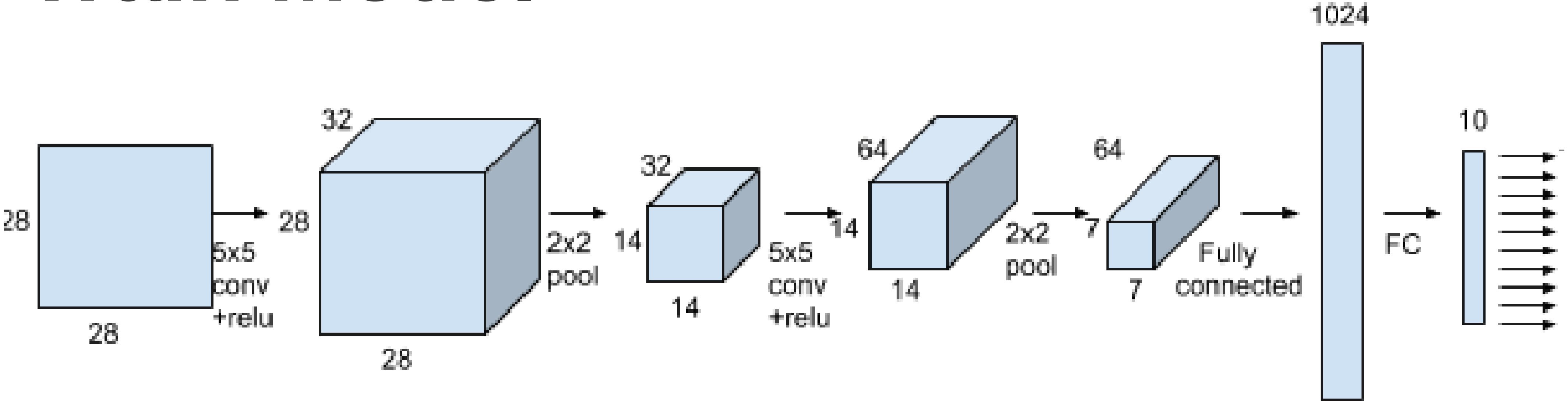


Figure D.2: Network architecture for MNIST classifier CNN

```
def call(self, x):
  x = self.conv1(x)
  x = self.maxpool1(x)
  x = self.conv2(x)
  x = self.maxpool2(x)
  x = self.flatten(x)
  x = self.d1(x)
  return self.d2(x)
```

https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D

tf.keras.layers.Conv2D(args, …)

- **Arguments:**
  - filters: Integer, the number of filters in the convolution.
  - kernel_size: An tuple/list of 2 integers, specifying the height and width of the 2D convolution window.
  - padding: One of "valid" or "same" (case-insensitive).
  - strides: An tuple/list of 2 integers, specifying the strides of the convolution along the height and width.
  - data_format: A string, one of channels_last (default) or channels_first.
    - channels_last corresponds to inputs with shape (batch, height, width, channels)
    - channels_first corresponds to inputs with shape (batch, channels, height, width).
  - activation: Activation function. Set it to None to maintain a linear activation.
  - name: A string, the name of the layer.

https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D

tf.keras.layers.MaxPooling2D(args, ...)

- **Arguments:**
  - inputs: Tensor input.
  - pool_size: An tuple/list of 2 integers: (pool_height, pool_width) specifying the size of the pooling window.
  - padding: One of "valid" or "same" (case-insensitive).
  - strides: An tuple/list of 2 integers, specifying the strides of the pooling operation.
  - data_format: A string, one of channels_last (default) or channels_first.
    - channels_last corresponds to inputs with shape (batch, height, width, channels)
    - channels_first corresponds to inputs with shape (batch, channels, height, width).
  - name: A string, the name of the layer.

https://www.tensorflow.org/versions/r1.15/api_docs/python/tf/layers/MaxPooling2D

tf.keras.layers.Flatten(args, …)

- **Arguments:**
  - data_format: A string, one of channels_last (default) or channels_first.
    - channels_last corresponds to inputs with shape (batch, height, width, channels)
    - channels_first corresponds to inputs with shape (batch, channels, height, width).

https://www.tensorflow.org/versions/r1.15/api_docs/python/tf/layers/Flatten

tf.keras.layers.Dropout(args, …)

- **Arguments:**
  - rate: The dropout rate, between 0 and 1. E.g. rate=0.1 would drop out 10% of input units.
  - name: The name of the layer (string).

Define Train Model

```
@tf.function
def train_step(images, labels):
  with tf.GradientTape() as tape:
    # training=True is only needed if there are layers with different
    # behavior during training versus inference (e.g. Dropout).
    predictions = model(images, training=True)
    loss = loss_object(labels, predictions)
  gradients = tape.gradient(loss, model.trainable_variables)
  optimizer.apply_gradients(zip(gradients, model.trainable_variables))
  train_loss(loss)
  train_accuracy(labels, predictions)
```

@tf.function

- tf.function 是一個decorator，任何經由@tf.function裝飾的function可以像原本一樣的被使用，但額外地獲得AutoGraph的效果。

tf.GradientTape()

Tape可以解釋為磁帶或膠帶，是TF裡面的context manager，用來關聯需要計算梯度的函數以及變數。
使用watch函數把需要計算梯度的變數x。
一般使用時，不用各別用watch 加入變數，直接用trainable_variables監控所有可訓練變數。

```
x = tf.constant(3.0)
with tf.GradientTape() as g:
  g.watch(x)
  y = x * x
dy_dx = g.gradient(y, x) # y’ = 2*x = 2*3 = 6
```

Define Test Model

```
@tf.function
def test_step(images, labels):
  # training=False is only needed if there are layers with different
  # behavior during training versus inference (e.g. Dropout).
  predictions = model(images, training=False)
  t_loss = loss_object(labels, predictions)

  test_loss(t_loss)
  test_accuracy(labels, predictions)
```

Training&Testing

```
EPOCHS = 5

for epoch in range(EPOCHS):

  # Reset the metrics at the start of the next epoch

  train_loss.reset_states()

  train_accuracy.reset_states()

  test_loss.reset_states()

  test_accuracy.reset_states()
```

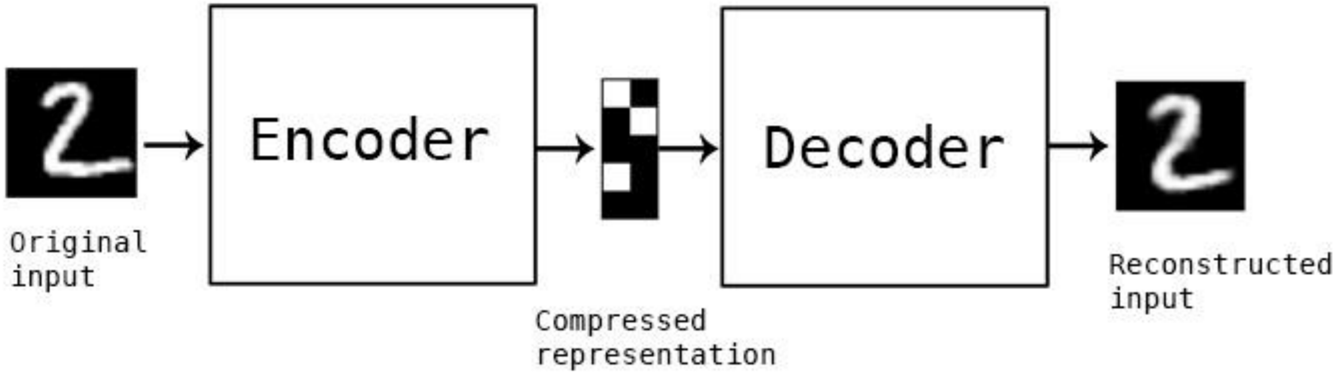
Training&Testing

```
for images, labels in train_ds:
  train_step(images, labels)

for test_images, test_labels in test_ds:
  test_step(test_images, test_labels)
```

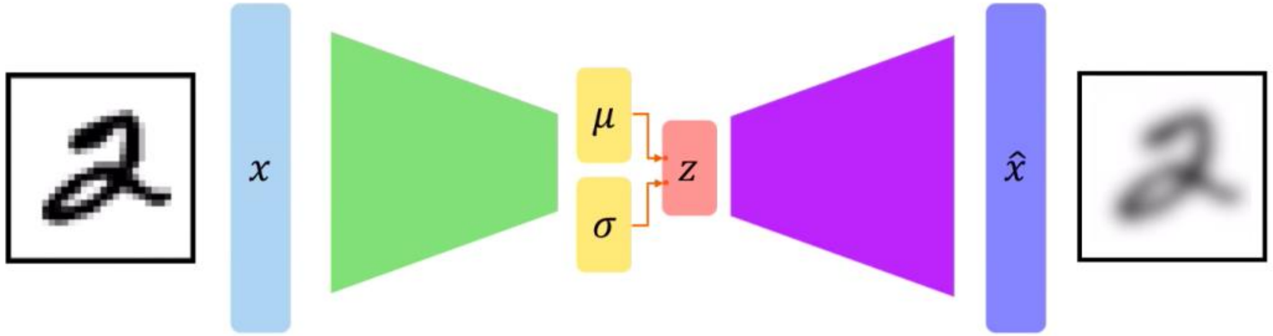
Training&Testing

```
print(
    f'Epoch {epoch + 1}, '
    f'Loss: {train_loss.result()}, '
    f'Accuracy: {train_accuracy.result() * 100}, '
    f'Test Loss: {test_loss.result()}, '
    f'Test Accuracy: {test_accuracy.result() * 100}'
 )
```

Generative Model : Variational Autoencocer

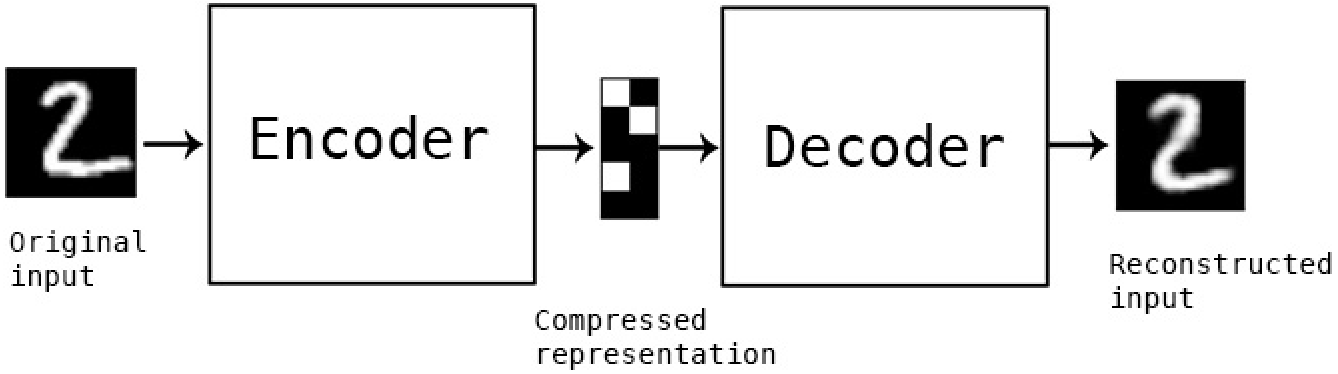


https://blog.keras.io/building-autoencoders-in-keras.html



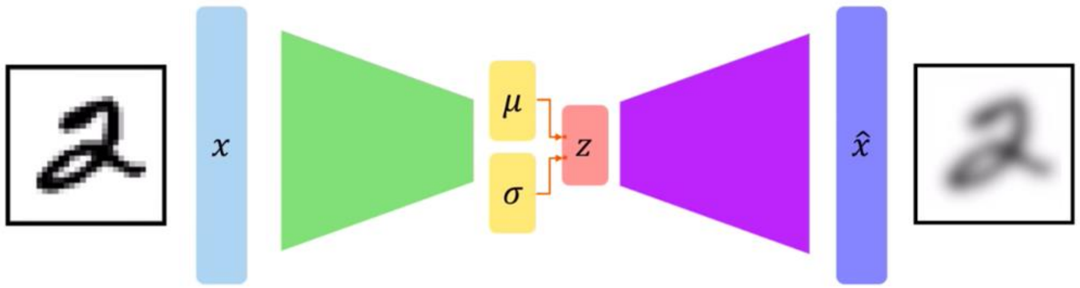
https://www.youtube.com/watch?v=rZufA635dq4

autoencoders



- 在autoencoder 實作裡，包含兩個部份1) Encoder 2) Decoder，Encoder 將 input 映射成 latent representation (可以想成是latent space中的一個向量)，而 Decoder 再將這個向量重新還原成原始的圖片。
- 用人腦運作來理解，看到一張圖片，我們只需要記住一些”特徵”、”概念”，而不需要把每一個像素都記下來，下次再看到同一張圖片仍能認得出來。
- 用壓縮的概念來理解 autoencoder， Encoder過程可以想成將一張圖片進行壓縮，用一個較低維度的 latent representation儲存原本圖片。
- 在壓縮時，資訊的完整性(破壞性壓縮Lossy Compression，非破壞性壓縮Lossless Compression)是與應用相關。
- 使用Autoencoder 的重點是latent representation 足以“有效的”代表原始資料嗎？Autoencoder可能只是死記硬背下這個 latent representation，當latent representation 存在微擾的話， Decoder 很有可能無法還原重建出圖片。

Variational Autoencoder (VAE)



- VAE 不再只是產生單一的latent representation來代表原來的輸入，而是產生一個Gaussain Distribution，並且提取這個分佈的平均值 (μ) 及標準差 (σ) 作為latent variable來代表。
- VAE 的 Decoder 再透過這些 latent variable 建構出原來的圖片
- 相較於autoencoder的latent向量， VAE用一個分佈(實際上是用μ和σ)來代表壓縮後的資訊，更能抵抗latent space的微擾。

Import module

```
!pip install tensorflow-probability
# to generate gifs
!pip install imageio
!pip install git+https://github.com/tensorflow/docs
from IPython import display
import glob
import imageio
import matplotlib.pyplot as plt
import numpy as np
import PIL
import tensorflow as tf
import tensorflow_probability as tfp
import time
```

讀資料並且進行前處理

```
(train_images, _), (test_images, _) = tf.keras.datasets.mnist.load_data()

def preprocess_images(images):
  images = images.reshape((images.shape[0], 28, 28, 1)) / 255.
  return np.where(images > .5, 1.0, 0.0).astype('float32')
# 模型採用Bernoulli distribution離散型機率分布(0-1分佈)
# 使用statically binarize 處理資料

train_images = preprocess_images(train_images)
test_images = preprocess_images(test_images)
# train_images.shape : (60000, 28, 28, 1)
# test_images.shape  : (10000, 28, 28, 1)
```

使用 tf.data.Dataset handle資料

```
train_size = 60000

batch_size = 32

test_size = 10000

"""## Use *tf.data* to batch and shuffle the data"""


train_dataset = (tf.data.Dataset.from_tensor_slices(train_images)
                 .shuffle(train_size).batch(batch_size))
test_dataset = (tf.data.Dataset.from_tensor_slices(test_images)
                .shuffle(test_size).batch(batch_size))
```

定義 Variational Autoencoder(Subclassing)

```
class CVAE(tf.keras.Model):
  """Convolutional variational autoencoder."""
  def __init__(self, latent_dim):
    super(CVAE, self).__init__()
    self.latent_dim = latent_dim
    self.encoder = tf.keras.Sequential(
...
     )
    self.decoder = tf.keras.Sequential(
...
    )
```

定義 Variational Autoencoder-1

```
self.encoder = tf.keras.Sequential(
    [
        tf.keras.layers.InputLayer(input_shape=(28, 28, 1)),
        tf.keras.layers.Conv2D(filters=32, kernel_size=3, strides=(2, 2), activation='relu'),
        tf.keras.layers.Conv2D(filters=64, kernel_size=3, strides=(2, 2), activation='relu'),
        tf.keras.layers.Flatten(),
        # No activation
        tf.keras.layers.Dense(latent_dim + latent_dim),
    ]
 )
```

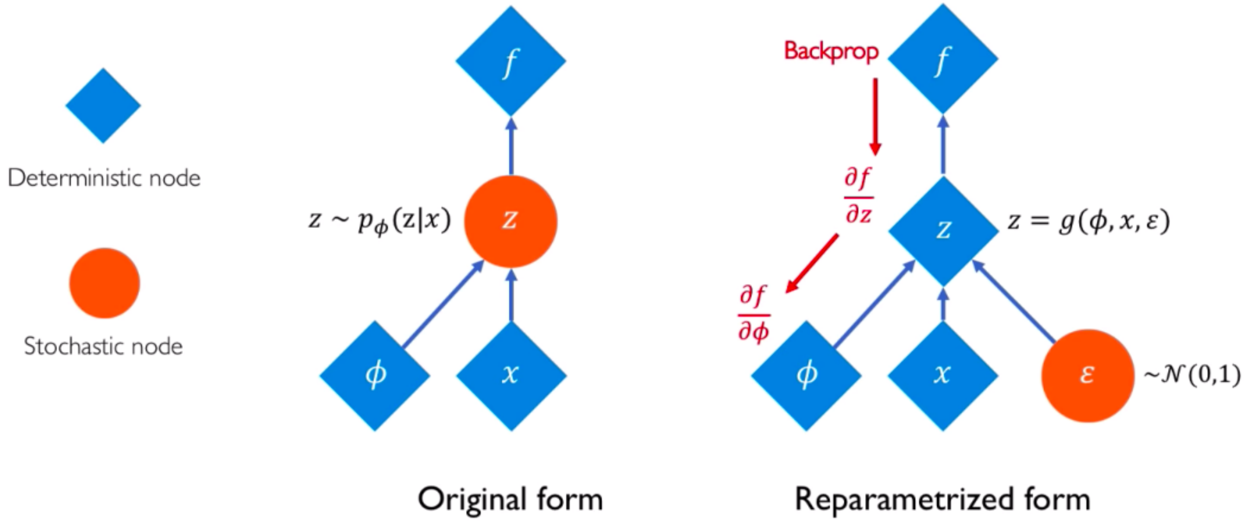
定義 Variational Autoencoder-2

```
self.decoder = tf.keras.Sequential(
    [
        tf.keras.layers.InputLayer(input_shape=(latent_dim,)),
        tf.keras.layers.Dense(units=7*7*32, activation=tf.nn.relu),
        tf.keras.layers.Reshape(target_shape=(7, 7, 32)),
        tf.keras.layers.Conv2DTranspose(filters=64, kernel_size=3, strides=2, padding='same',activation='relu'),
        tf.keras.layers.Conv2DTranspose(filters=32, kernel_size=3, strides=2, padding='same',activation='relu'),
        # No activation
        tf.keras.layers.Conv2DTranspose(filters=1, kernel_size=3, strides=1, padding='same'),
    ]
  )
```

定義 Variational Autoencoder-3

- Encoder : 將 input 圖片映射成 latent representation z
  - input : x (圖片)
  - output: Gaussian Distribution 的 μ 和 σ 。
- Sample
  - 從原本分佈Gaussain中抽出隨機 ε 。
- Decoder : 將 latent representation z 作為 input，output 出 image
  - input : z (從 Gaussian Distribution 中 sample 值出來)
  - output : x (圖片)
- Reparameterization
  - 使用Encoder output的 μ 和 σ 以及 ，來代表這個 latent representation z

$$z = \mu + \sigma \odot \epsilon$$



定義 Variational Autoencoder-4

```
@tf.function
 def sample(self, eps=None):
  if eps is None:
   eps = tf.random.normal(shape=(100, self.latent_dim))
  return self.decode(eps, apply_sigmoid=True)

 def encode(self, x):
  mean, logvar = tf.split(self.encoder(x), num_or_size_splits=2, axis=1)
  return mean, logvar
```

定義 Variational Autoencoder-5

```
def reparameterize(self, mean, logvar):
  eps = tf.random.normal(shape=mean.shape)
  return eps * tf.exp(logvar * .5) + mean

def decode(self, z, apply_sigmoid=False):
  logits = self.decoder(z)
  if apply_sigmoid:
    probs = tf.sigmoid(logits)
    return probs
  return logits
```

定義 Loss Function

- VAE 想做到的是最大化 ELBO(evidence lower bound)

$$\log \mathrm{p}\,(\mathrm{x}) \geq ELBO = \mathrm{E}_{\mathrm{q}(\mathrm{z}|\mathrm{x})}\left[\log \frac{p(x,z)}{q(z|x)}\right].$$

- 實作上則是用optimize the single sample Monte Carlo estimate of this expectation

$$\log \mathrm{p}\,(\mathrm{x}|\mathrm{z}) + \log \mathrm{p}\,(\mathrm{z}) - \log \mathrm{q}\,(\mathrm{z}|\mathrm{x}),$$

- 最簡單的也可以直接使用Kullback-Leibler Divergence(KL term,相對熵relative entropy)比較兩個分佈的差異。

$$D_{\mathrm{KL}}(P\|Q) = \sum_i P(i) \ln \frac{P(i)}{Q(i)}.$$

定義 Loss Function-1

```
optimizer = tf.keras.optimizers.Adam(1e-4)


def log_normal_pdf(sample, mean, logvar, raxis=1):
  log2pi = tf.math.log(2. * np.pi)
  return tf.reduce_sum(
      -.5 * ((sample - mean) ** 2. * tf.exp(-logvar) + logvar + log2pi),
      axis=raxis)
```

定義 Loss Function-2

```
def compute_loss(model, x):
  mean, logvar = model.encode(x)
  z = model.reparameterize(mean, logvar)
  x_logit = model.decode(z)
  cross_ent = tf.nn.sigmoid_cross_entropy_with_logits(logits=x_logit, labels=x)
  logpx_z = -tf.reduce_sum(cross_ent, axis=[1, 2, 3])
  logpz = log_normal_pdf(z, 0., 0.)
  logqz_x = log_normal_pdf(z, mean, logvar)
  return -tf.reduce_mean(logpx_z + logpz - logqz_x)
```

Define Train Model-3

```
@tf.function
def train_step(model, x, optimizer):
  """Executes one training step and returns the loss.

  This function computes the loss and gradients, and uses the latter to
  update the model's parameters.
  """
  with tf.GradientTape() as tape:
    loss = compute_loss(model, x)
  gradients = tape.gradient(loss, model.trainable_variables)
  optimizer.apply_gradients(zip(gradients, model.trainable_variables))
```

Training&Testing-1

```
epochs = 10
# set the dimensionality of the latent space to a plane for visualization later
latent_dim = 2
num_examples_to_generate = 16

# keeping the random vector constant for generation (prediction) so
# it will be easier to see the improvement.
random_vector_for_generation = tf.random.normal(
    shape=[num_examples_to_generate, latent_dim])
model = CVAE(latent_dim)
```

Training&Testing-2

```
for epoch in range(1, epochs + 1):
  start_time = time.time()
  for train_x in train_dataset:
    train_step(model, train_x, optimizer)
  end_time = time.time()

  loss = tf.keras.metrics.Mean()
```

Training&Testing-3

```
for test_x in test_dataset:
  loss(compute_loss(model, test_x))
elbo = -loss.result()
display.clear_output(wait=False)
print('Epoch: {}, Test set ELBO: {}, time elapse for current epoch: {}'
      .format(epoch, elbo, end_time - start_time))
```