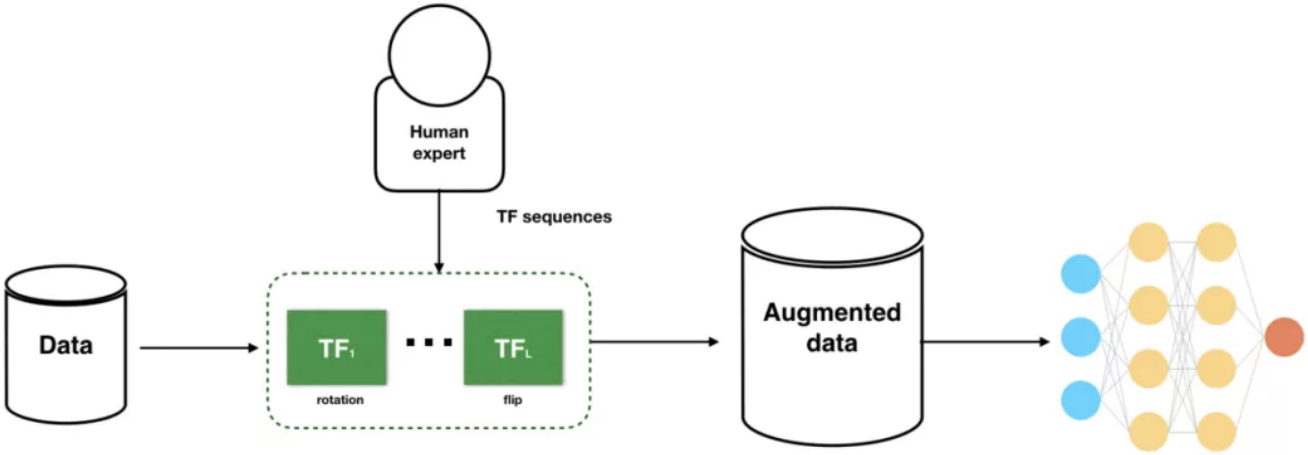
一文歸納Ai數據增強之法

原創 算法進階 算法進階 3月17日

數據、算法、算力是人工智能發展的三要素。數據決定了Ai模型學習的上限，數據規模越大、質量越高，模型就能夠擁有更好的泛化能力。然而在實際工程中，經常有數據量太少(相對模型而言)、樣本不均衡、很難覆蓋全部的場景等問題，解決這類問題的一個有效途徑是通過數據增強（Data Augmentation），使模型學習獲得較好的泛化性能。



1 數據增強介紹

數據增強（Data Augmentation）是在不實質性的增加數據的情況下，從原始數據加工出更多的表示，提高原數據的數量及質量，以接近於更多數據量產生的價值。其原理是，**通過對原始數據融入先驗知識，加工出更多數據的表示，有助於模型判別數據中統計噪聲**，加強本體特徵的學習，減少模型過擬合，提升泛化能力。

> 如經典的機器學習例子--哈士奇誤分類為狼：通過可解釋性方法，可發現錯誤分類是由於圖像上的雪造成的。通常狗對比狼的圖像裡面雪地背景比較少，分類器學會使用雪作為一個特徵來將圖像分類為狼還是狗，而忽略了動物本體的特徵。此時，可以通過數據增強的方法，增加變換後的數據(如背景換色、加入噪聲等方式)來訓練模型，幫助模型學習到本體的特徵，提高泛化能力。
>
> 
>
> 需要關注的是，數據增強樣本也有可能是引入片面噪聲，導致過擬合。此時需要考慮的是調整數據增強方法，或者通過算法(可藉鑑Pu-Learning思路)選擇增強數據的最佳子集，以提高模型的泛化能力。

常用數據增強方法可分為：**基於樣本變換的數據增強及基於深度學習的數據增強。**

2 基於樣本變換的數據增強

樣本變換數據增強即採用預設的數據變換規則進行已有數據的擴增，包含單樣本數據增強和多樣本數據增強。

2.1 單樣本增強

單(圖像)樣本增強主要有幾何操作、顏色變換、隨機擦除、添加噪聲等方法，可參見imgaug開源庫。

2.2 多樣本數據增強方法

多樣本增強是通過先驗知識組合及轉換多個樣本，主要有Smote、SamplePairing、Mixup等方法在特徵空間內構造已知樣本的鄰域值。

- Smote

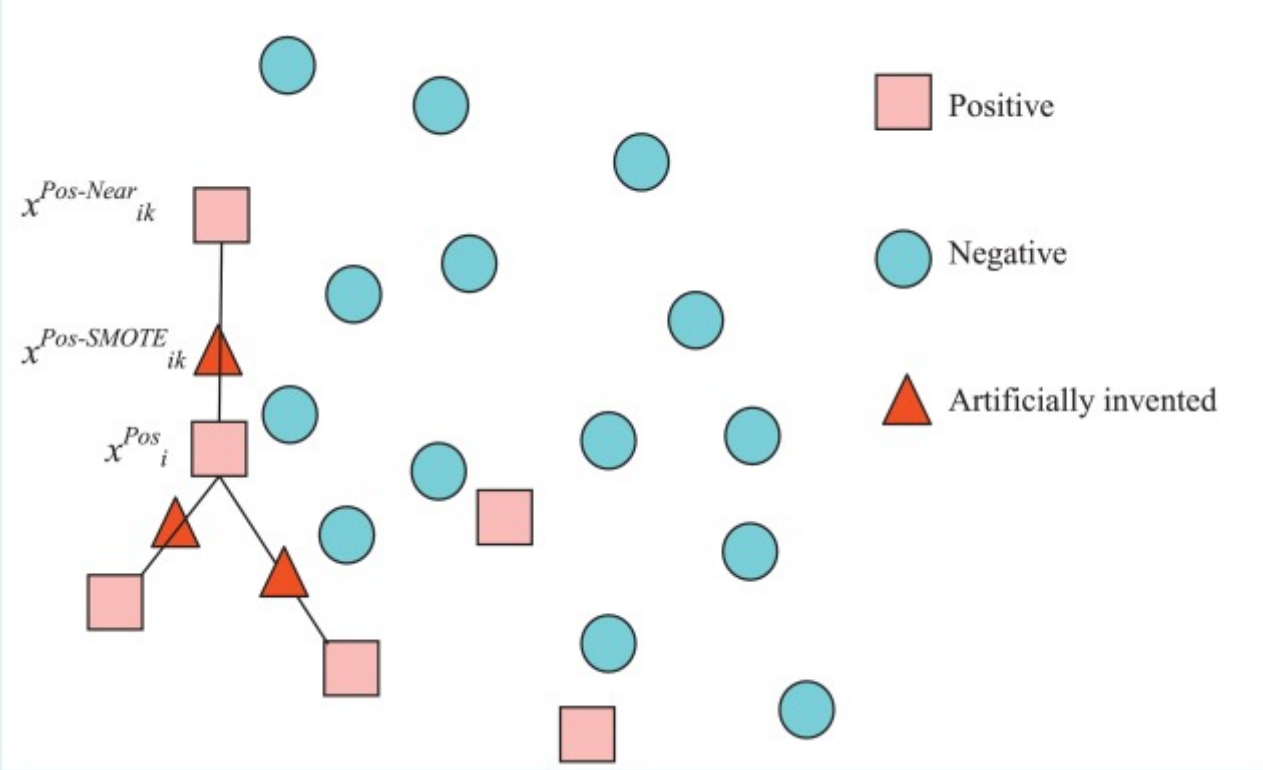
Smote (Synthetic Minority Over-sampling Technique)方法較常用於樣本均衡學習，核心思想是從訓練集隨機同類的兩近鄰樣本合成一個新的樣本，其方法可以分為三步：

1、 對於各樣本X_i，計算與同類樣本的歐式距離，確定其同類的K個(如圖3個)近鄰樣本；

2、從該樣本k近鄰中隨機選擇一個樣本如近鄰X_ik，生成新的樣本:

```
Xsmote_ik =  Xi  +  rand(0,1) * |X_i - X_ik|
```

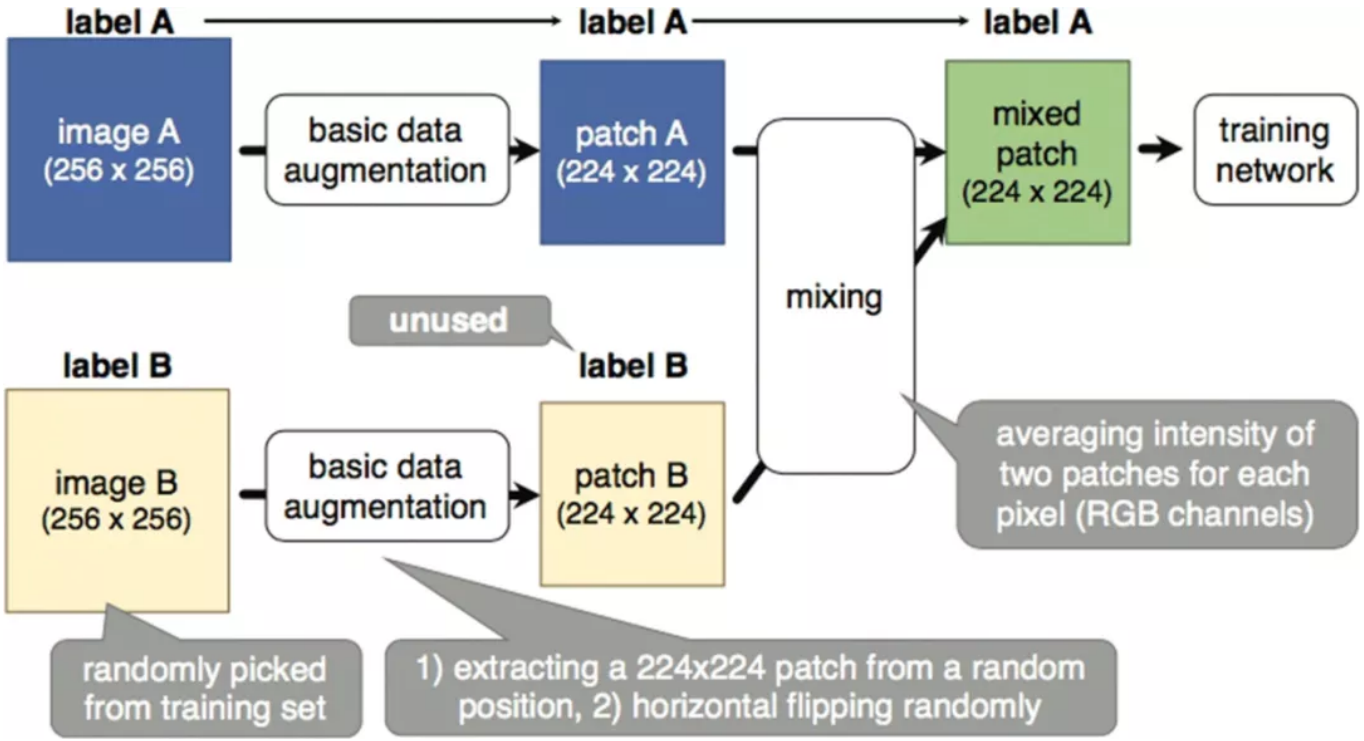
3、重複2步驟迭代N次，可以合成N個新的樣本。




```
# SMOTE
from imblearn.over_sampling import SMOTE

print("Before OverSampling, counts of label\n{}".format(y_train.value_counts()))
smote = SMOTE()
x_train_res, y_train_res = smote.fit_resample(x_train, y_train)
print("After OverSampling, counts of label\n{}".format(y_train_res.value_counts()))
```

- SamplePairing

SamplePairing算法的核心思想是從訓練集隨機抽取的兩幅圖像疊加合成一個新的樣本（像素取平均值），使用第一幅圖像的label作為合成圖像的正確label。



- Mixup

Mixup算法的核心思想是按一定的比例隨機混合兩個訓練樣本及其標籤，這種混合方式不僅能夠增加樣本的多樣性，且能夠使決策邊界更加平滑，增強了難例樣本的識別，模型的魯棒性得到提升。其方法可以分為兩步：

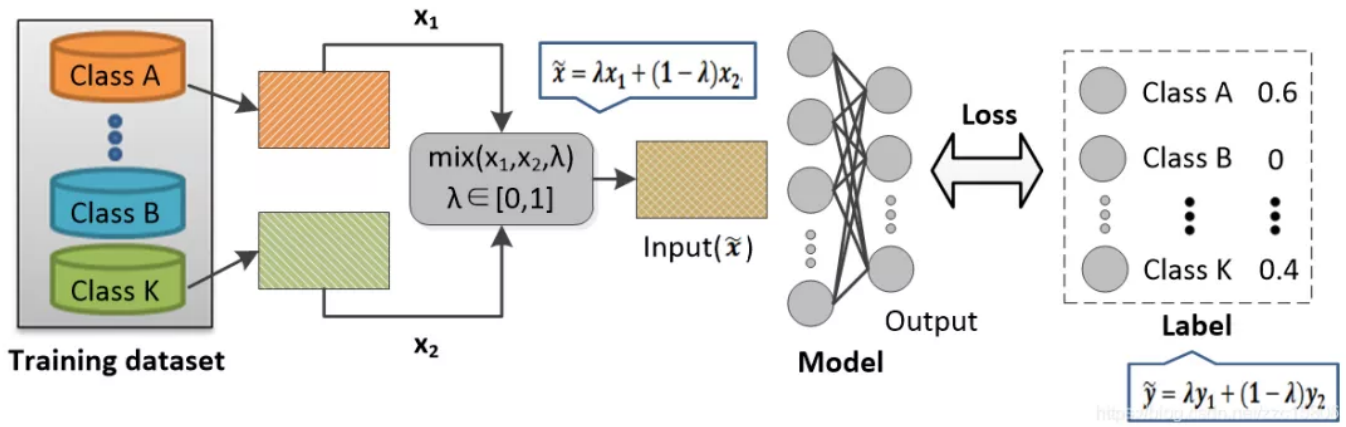
1、從原始訓練數據中隨機選取的兩個樣本(xi, yi) and (xj, yj)。其中y(原始label)用one-hot 編碼。

2、對兩個樣本按比例組合，形成新的樣本和帶權重的標籤

```
x˜ = λxi + (1 − λ)xj
```

```
y^ = λy1 + (1 − λ)yj
```

最終的loss為各標籤上分別計算cross-entropy loss，加權求和。其中λ ∈ [0, 1]， λ是mixup的超參數，控制兩個樣本插值的強度。



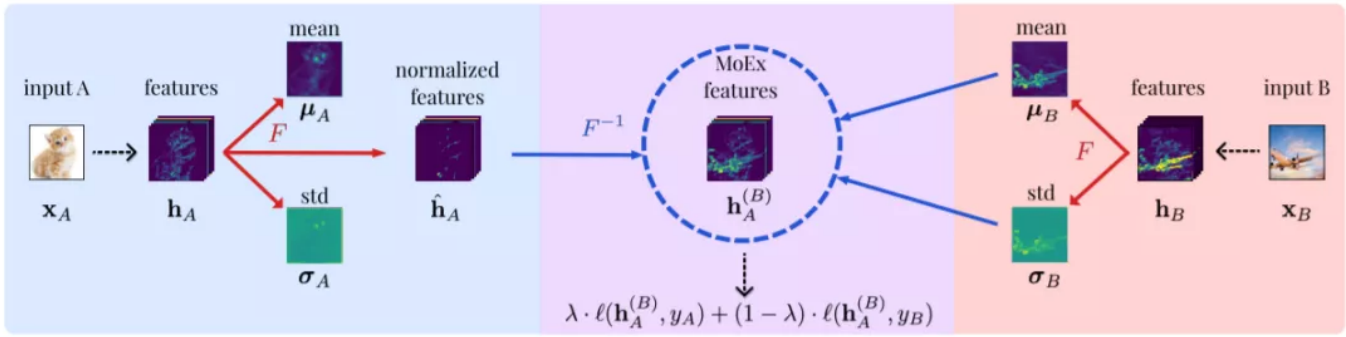

```python
# Mixup
def mixup_batch(x, y, step, batch_size, alpha=0.2):
    """
    get batch data
    :param x: training data
    :param y: one-hot label
    :param step: step
    :param batch_size: batch size
    :param alpha: hyper-parameter α, default as 0.2
    :return:  x y
    """
    candidates_data, candidates_label = x, y
    offset = (step * batch_size) % (candidates_data.shape[0] - batch_size)

    # get batch data
```

```
    train_features_batch = candidates_data[offset:(offset + batch_size)]
    train_labels_batch = candidates_label[offset:(offset + batch_size)]

    if alpha == 0:
        return train_features_batch, train_labels_batch

    if alpha > 0:
        weight = np.random.beta(alpha, alpha, batch_size)
        x_weight = weight.reshape(batch_size, 1)
        y_weight = weight.reshape(batch_size, 1)
        index = np.random.permutation(batch_size)
        x1, x2 = train_features_batch, train_features_batch[index]
        x = x1 * x_weight + x2 * (1 - x_weight)
        y1, y2 = train_labels_batch, train_labels_batch[index]
        y = y1 * y_weight + y2 * (1 - y_weight)
        return x, y
```

3 基於深度學習的數據增強

3.1 特徵空間的數據增強

不同於傳統在輸入空間變換的數據增強方法，神經網絡可將輸入樣本映射為網絡層的低維向量(表徵學習)，從而直接在學習的特徵空間進行組合變換等進行數據增強，如MoEx方法等。

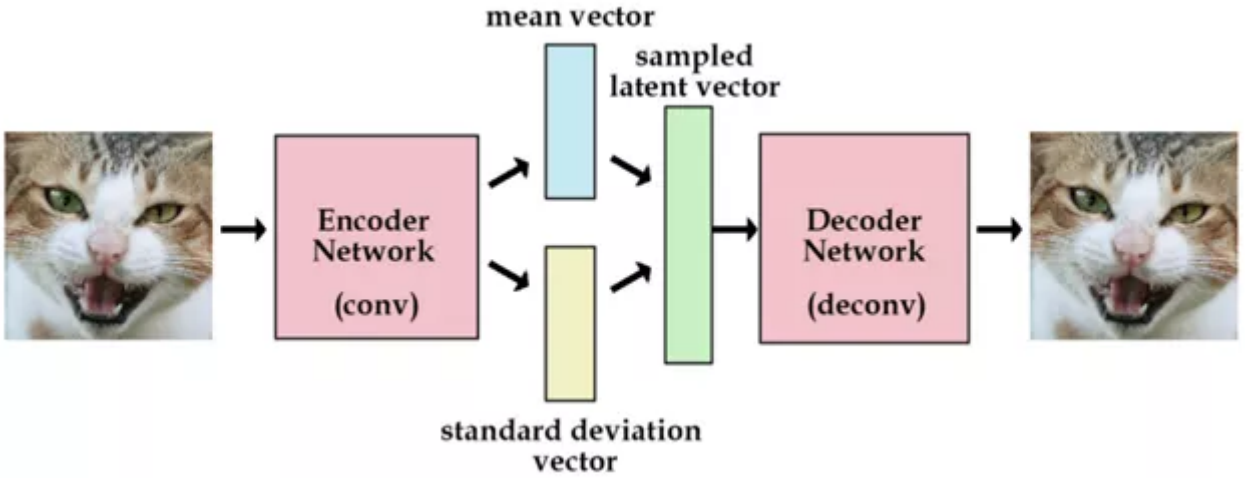


3.2 基於生成模型的數據增強

生成模型如變分自編碼網絡(Variational Auto-Encoding network, VAE)和生成對抗網絡(Generative Adversarial Network, GAN)，其生成樣本的方法也可以用於數據增強。這種基於網絡合成的方法相比於傳統的數據增強技術雖然過程更加複雜, 但是生成的樣本更加多樣。

- 變分自編碼器VAE

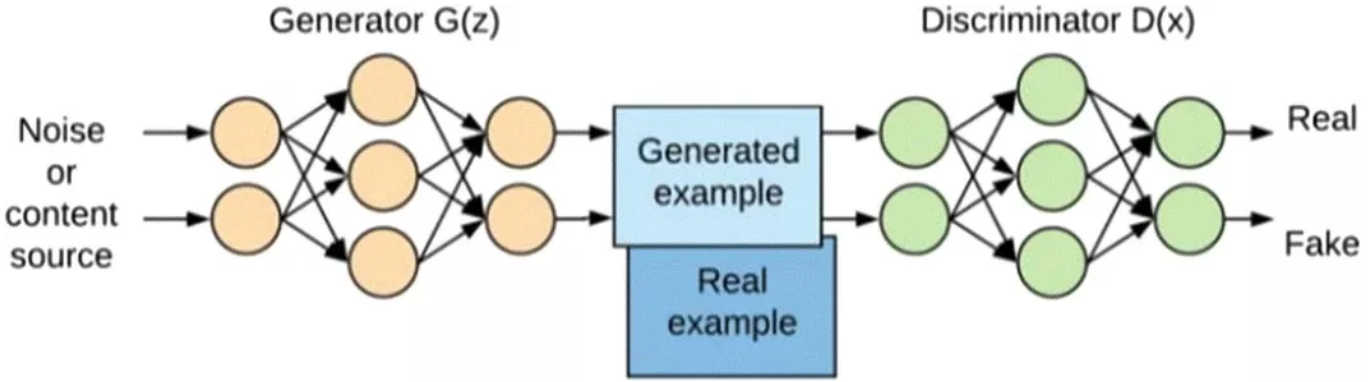
  變分自編碼器（Variational Autoencoder，VAE）其基本思路是：將真實樣本通過編碼器網絡變換成一個理想的數據分佈，然後把數據分佈再傳遞給解碼器網絡，構造出生成樣本，模型訓練學習的過程是使生成樣本與真實樣本足夠接近。



```
# VAE模型
class VAE(keras.Model):
    ...
    def train_step(self, data):
        with tf.GradientTape() as tape:
            z_mean, z_log_var, z = self.encoder(data)
            reconstruction = self.decoder(z)
            reconstruction_loss = tf.reduce_mean(
                tf.reduce_sum(
                    keras.losses.binary_crossentropy(data, reconstruction), axis=(1, 2)
                )
            )
            kl_loss = -0.5 * (1 + z_log_var - tf.square(z_mean) - tf.exp(z_log_var))
            kl_loss = tf.reduce_mean(tf.reduce_sum(kl_loss, axis=1))
            total_loss = reconstruction_loss + kl_loss
        grads = tape.gradient(total_loss, self.trainable_weights)
        self.optimizer.apply_gradients(zip(grads, self.trainable_weights))
        self.total_loss_tracker.update_state(total_loss)
        self.reconstruction_loss_tracker.update_state(reconstruction_loss)
        self.kl_loss_tracker.update_state(kl_loss)
        return {
            "loss": self.total_loss_tracker.result(),
            "reconstruction_loss": self.reconstruction_loss_tracker.result(),
            "kl_loss": self.kl_loss_tracker.result(),
        }
```

- 生成對抗網絡GAN

  生成對抗網絡-GAN(Generative Adversarial Network)由生成網絡(Generator, G)和判別網絡(Discriminator, D)兩部分組成，生成網絡構成一個映射函數$G: Z \to X$（輸入噪聲z,輸出生成的圖像數據x）,判別網絡判別輸入是來自真實數據還是生成網絡生成的數據。

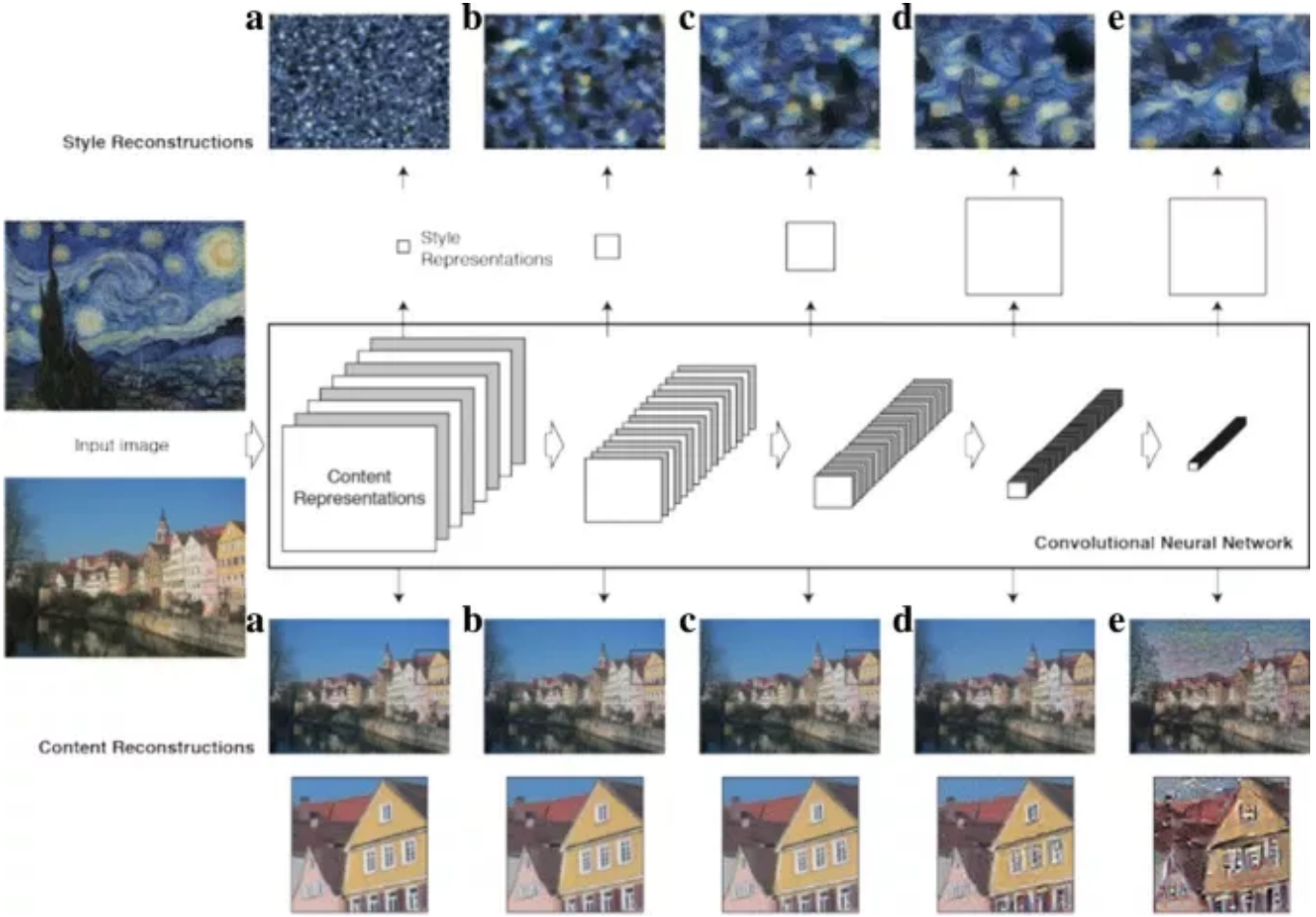



```
# DCGAN模型

class GAN(keras.Model):
    ...
    def train_step(self, real_images):
        batch_size = tf.shape(real_images)[0]
        random_latent_vectors = tf.random.normal(shape=(batch_size, self.latent_dim))
        # G: Z→X（输入噪声z，输出生成的图像数据x）
        generated_images = self.generator(random_latent_vectors)
        # 合并生成及真实的样本并赋判定的标签
        combined_images = tf.concat([generated_images, real_images], axis=0)
        labels = tf.concat(
            [tf.ones((batch_size, 1)), tf.zeros((batch_size, 1))], axis=0
        )
        # 标签加入随机噪声
        labels += 0.05 * tf.random.uniform(tf.shape(labels))
        # 训练判定网络
        with tf.GradientTape() as tape:
            predictions = self.discriminator(combined_images)
            d_loss = self.loss_fn(labels, predictions)
        grads = tape.gradient(d_loss, self.discriminator.trainable_weights)
        self.d_optimizer.apply_gradients(
            zip(grads, self.discriminator.trainable_weights)
        )
```

```
        )

        random_latent_vectors = tf.random.normal(shape=(batch_size, self.latent_dim))
        # 赋生成网络样本的标签(都赋为真实样本)
        misleading_labels = tf.zeros((batch_size, 1))
        # 训练生成网络
        with tf.GradientTape() as tape:
            predictions = self.discriminator(self.generator(random_latent_vectors))
            g_loss = self.loss_fn(misleading_labels, predictions)
        grads = tape.gradient(g_loss, self.generator.trainable_weights)
        self.g_optimizer.apply_gradients(zip(grads, self.generator.trainable_weights))
        # 更新损失
        self.d_loss_metric.update_state(d_loss)
        self.g_loss_metric.update_state(g_loss)
        return {
            "d_loss": self.d_loss_metric.result(),
            "g_loss": self.g_loss_metric.result(),
        }
```

3.3 基於神經風格遷移的數據增強

神經風格遷移(Neural Style Transfer)可以在保留原始內容的同時，將一個圖像的樣式轉移到另一個圖像上。除了實現類似色彩空間照明轉換，還可以生成不同的紋理和藝術風格。



神經風格遷移是通過優化三類的損失來實現的：

style_loss：使生成的圖像接近樣式參考圖像的局部紋理；

content_loss：使生成的圖像的內容表示接近於基本圖像的表示；

total_variation_loss：是一個正則化損失，它使生成的圖像保持局部一致。

```
# 样式损失
```

```
def style_loss(style, combination):
    S = gram_matrix(style)
    C = gram_matrix(combination)
    channels = 3
    size = img_nrows * img_ncols
    return tf.reduce_sum(tf.square(S - C)) / (4.0 * (channels ** 2) * (size ** 2))

# 内容损失
def content_loss(base, combination):
    return tf.reduce_sum(tf.square(combination - base))

# 正则损失
def total_variation_loss(x):
    a = tf.square(
        x[:, : img_nrows - 1, : img_ncols - 1, :] - x[:, 1:, : img_ncols - 1, :]
    )
    b = tf.square(
        x[:, : img_nrows - 1, : img_ncols - 1, :] - x[:, : img_nrows - 1, 1:, :]
    )
    return tf.reduce_sum(tf.pow(a + b, 1.25))
```

3.4 基於元學習的數據增強

深度學習研究中的元學習(Meta learning)通常是指使用神經網絡優化神經網絡，元學習的數據增強有神經增強(Neural augmentation)等方法。

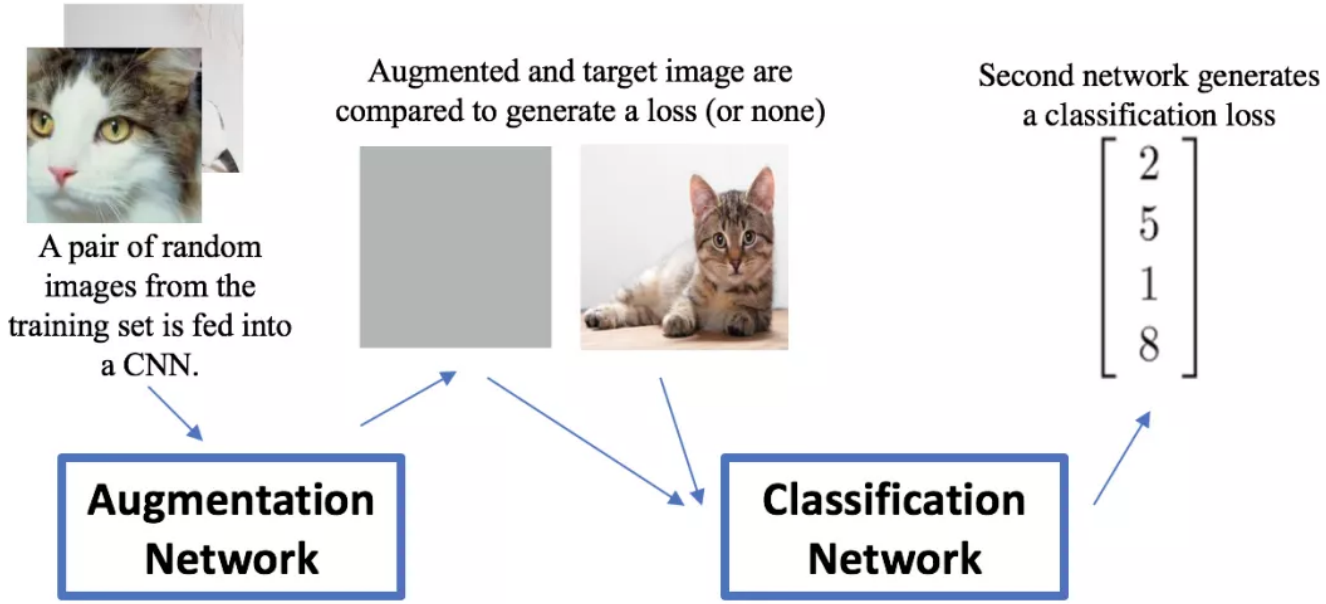
- 神經增強

神經增強(Neural augmentation)是通過神經網絡組的學習以獲得較優的數據增強並改善分類效果的一種方法。其方法步驟如下：

1、獲取與target圖像同一類別的一對隨機圖像，前置的增強網絡通過CNN將它們映射為合成圖像，合成圖像與target圖像對比計算損失；

2、將合成圖像與target圖像神經風格轉換後輸入到分類網絡中，並輸出該圖像分類損失；

3、將增強與分類的loss加權平均後，反向傳播以更新分類網絡及增強網絡權重。使得其輸出圖像的同類內差距減小且分類準確。



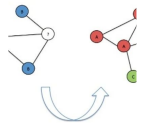
算法進階公眾號**閱讀原文**可訪問GitHub源碼

閱讀原文 文章已於2021/03/18修改

喜歡此內容的人還喜歡

半監督算法概覽(Python)

算法進階



江蘇：12部門聯合，首屆“民法典宣傳月”活動已啟動！

中國普法

從雅魯藏布大峽谷穿過，一條公路能有多美？

共青團中央