과목명: 시스템프로그래밍

담당교수명 : 소정민

<<Assignment 5>>

서강대학교 컴퓨터공학과

이정원20171672

목차

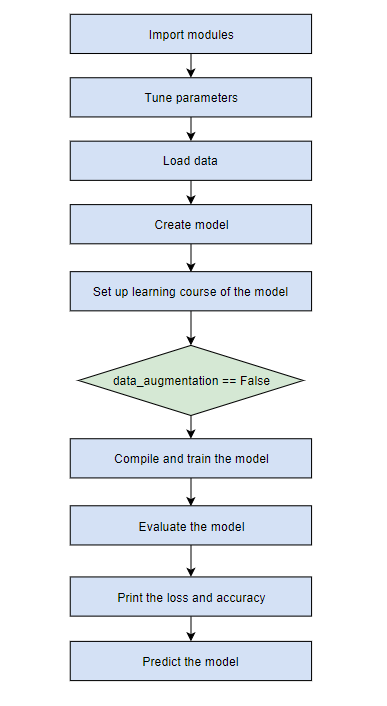
1. **프로그램 개요3**
2. **프로그램 설명3**
   1. 프로그램 흐름도3
3. 모듈 정의**5**
4. **코드9**
5. **프로그램 개요**

이 프로젝트에서는 머신러닝에 대한 이해도를 높이는 것을 목적으로 한다.

1. **프로그램 설명**
   1. **프로그램 흐름도**

CIFAR-10 데이터셋은 10개의 클래스로 이루어져 있으며, 1개의 클래스 당 6000개의 32\*32 컬러 이미지로 이루어져 있다. 총 60000개의 이미지이며, 그 중 50000개는 training 이미지들이고, 10000개는 test 이미지들이다. 10개의 클래스는 airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck 이 있다. 이 클래스들은 서로 전혀 겹치지 않는 것이 특징이다.

이 프로그램은 CIFAR-10 데이터셋 분류의 정확도가 높은 것을 우선시하여, 정확도를 높이기 위해 코드를 작성하였다. 우선 전체적인 코드의 흐름도는 다음과 같다.



* **Import modules**
* 가장 처음 필요한 모듈들을 import 한다.
* **Tune parameters**
* 코드에서 필요한 변수들을 선언한다.
* batch\_size : 한 번의 batch마다 주는 데이터 샘플의 크기를 뜻한다.
* num\_classes : 이미지 분류할 클래스의 개수이다. 여기서는 10이다.
* epochs : 전체 데이터셋에 대해 몇 번을 학습할 것인지 결정하는 변수를 뜻한다.
* data\_augmentation : 실시간 data augmentation을 실행할 것인지 말 것인지 결정하는 변수를 뜻한다.
* num\_predictions : prediction의 개수를 뜻한다.
* save\_dir : 모델을 저장할 디렉토리를 준비한다는 것을 뜻한다.
* model\_name : 모델의 이름을 뜻한다.
* **Load data**
* 데이터를 train set 과 test set으로 나누고, 클래스 벡터를 이진 클래스 행렬로 변환시킨다.
* **Create model**
* 모델을 생성할 때, 모델을 순차적으로 쌓아 만들겠다는 뜻으로 “model = Sequential() “ 과 같이 선언한다.
* model.add 를 통해 모델에 레이어들을 쌓는 과정을 거친다. 따라서 모델이 구성되게 된다.
* **Set up learning course of the model**
* 모델의 학습과정을 설정하는 단계이다. 모델을 정의한 후 모델을 최적화 알고리즘으로 엮어 보는 것이다.
* **Compile and train the model**
* 모델을 컴파일 할 때, “model.compile(loss='categorical\_crossentropy', optimizer=opt, metrics=['accuracy'])” 에서 loss 는 현재 가중치 세트를 평가하는 데 사용한 손실 함수이며, 다중 클래스 문제이므로 ‘categorical\_crossentropy’ 으로 지정한다. optimizer은 바로 전 단계에서 학습과정을 설정한 변수 opt로 지정하고, metrics는 평가 척도를 나타내는데 분류 문제에서는 일반적으로 ‘accuracy’를 지정한다.
* 모델을 학습시킬 때, data augmentation을 사용하지 않을 경우 fit() 함수를 쓰고, 사용할 경우 제너레이터로 생성된 배치로 학습시키는 fit\_generator() 함수를 쓴다. 후자일 때 ImageDataGenerator라는 제너레이터로 이미지를 담고 있는 배치로 학습시키기 위해 우선 ImageDataGenerator 클래스를 이용하여 객체를 생성하여 제너레이터를 생성한다. 이후 fit\_generator() 함수를 사용한다. 이 함수의 첫 번째 인자는 훈련 데이터셋을 제공할 제너레이터 (datagen)을 지정하고, steps\_per\_epoch는 한 epoch에 사용할 스텝 수를 지정한다. epochs는 초반부에서 지정한 epochs로 지정하고, validation\_data는 검증 데이터셋을 제공할 제너레이터를 지정한다. 예를 들면, 퀴즈를 보는데 성적에는 안 들어가는 것인데, 잘 학습시키는지 확인하기 위해 쓰이는 변수이다. Data Augmentation을 쓸지 안 쓸지에 대한 여부는 변수 data\_augmentation을 True 혹은 False로 바꿔 주면 된다. (쓴다면 True, 안 쓸 것이라면 False) 여기서는 쓰지 않을 것이라고 선언하였다.
* **Evaluate the model**
* evaluate() 함수를 이용하여 학습한 모델을 평가한다.
* **Print the loss and accuracy**
* 손실과 정확도를 출력한다.
* **Predict the model**
* 모델을 predict() 함수를 이용해 사용해 본다. plot image 도 출력해 본다.

1. **모듈 정의**

더 높은 정확도를 높이기 위해 다음과 같은 방법을 사용하였다.

* **Increase the epoch**
* epoch 수를 40으로 지정하였다. 이를 시험하기 위해 다양한 epoch를 시도해 봤는데, 확실히 epoch 이 낮을 때보다 높을 때 높은 정확도가 나왔다.
* **Stack more layers**
* Core 레이어, Convolution 레이어, Pooling 레이어를 이용하여 기존 레이어보다 더 많은 레이어를 쌓는 것이 정확도를 높이는 데 높은 기여를 하였다. 레이어들의 종류는 다음과 같다.
* 케라스에서 제공되는 필터로 특징을 뽑아 주는 컨볼루션 레이어 중 Conv2D 클래스를 이용하였는데, 주요 인자는 다음과 같다.

ⓐ 첫 번째 인자는 컨볼루션 필터의 수이다.

ⓑ 두 번째 인자는 컨볼루션 커널의 (행, 열)이다.

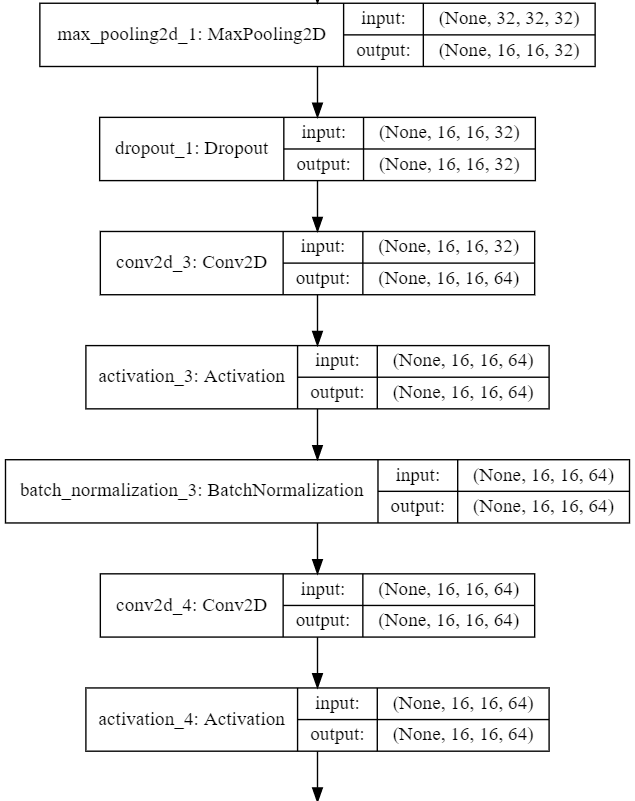
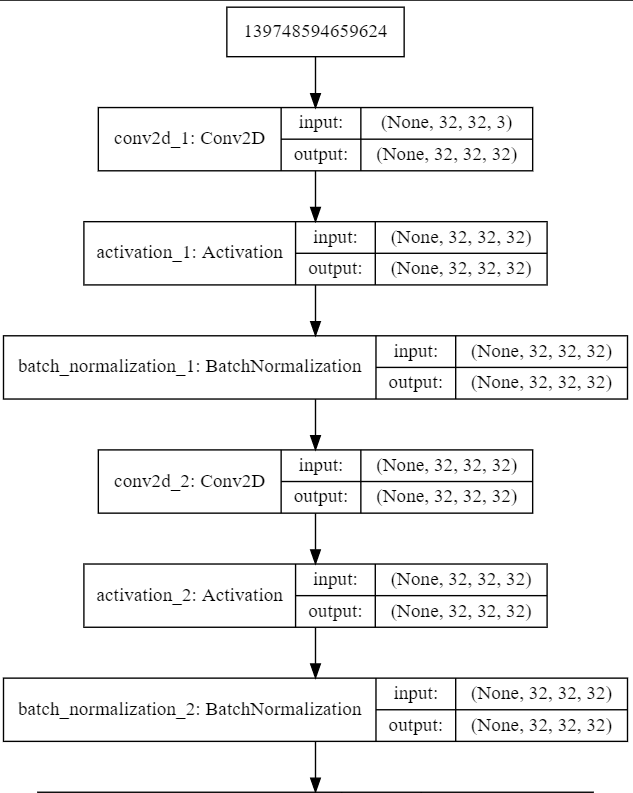
ⓒ padding은 경계 처리 방법을 정의하는데, 출력 이미지 사이즈가 입력 이미지 사이즈와 동일하다는 뜻으로 ‘same’으로 정의하였다.

ⓓ kernel\_regularizer은 Overfitting을 완화시키기 위해 가중치가 작은 값을 가지도록 네트워크의 복잡도에 제약을 가하는 역할을 한다. 이는 가중치 값의 분포를 좀 더 균일하게 만들어 주는데, “가중치 규제 (weight regularization)’ 이라고 불린다. 네트워크의 손실 함수에 큰 가중치에 해당하는 비용을 추가하는데, 이 비용에는 L1 규제와 L2 규제가 있다. 여기서는 가중치의 제곱에 비례하는 비용이 추가되는 L2 규제를 사용하였다. 신경망에서는 L2 규제를 가중치 감쇠(weight decay)라고도 부른다. (L1 규제보다 L2 규제가 Overfitting에 훨씬 잘 견디는 특징이 있다.)

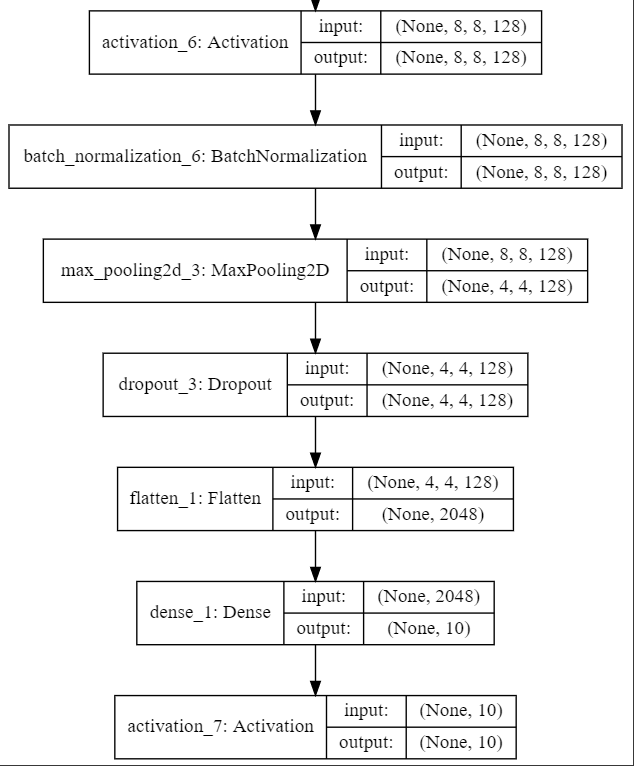
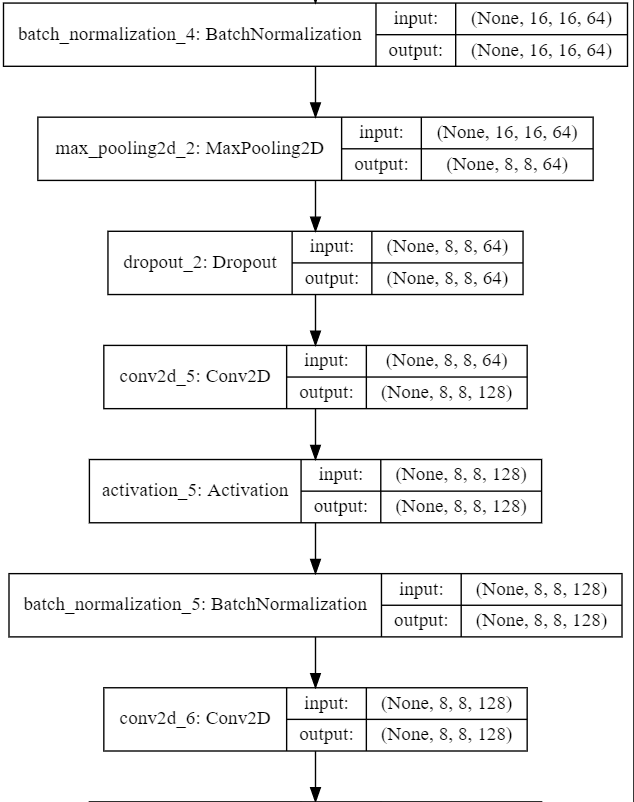
ⓔ input\_shape은 샘플 수를 제외한 입력 형태를 정의한다. 모델의 첫 레이어일 때만 정의하며 된다.

ⓕ activation은 활성화 함수를 설정해 주는 것이다. 마지막 레이어를 제외한 은닉층 레이어에는 rectifier 함수인 ‘relu’를 사용하였고, 출력층인 마지막 레이어에는 ‘softmax’를 사용하였다.

* 맥스 풀링 (Max Pooling)레이어는 사소한 변화를 무시해 준다. pool\_size를 통해 수직, 수평 축소 비율을 지정한다. 여기서 (2,2)로 지정해 주었고, 출력 영상 크기가 입력 영상 크기의 반으로 줄어들게 된다. 예를 들면, 차마다 바퀴의 위치가 조금씩 다른데 이러한 차이가 차라고 인식하는 데 있어서는 큰 영향을 미치지 않게 하는 것이다.
* 플래튼 (Flatten) 레이어는 영상을 일차원으로 바꿔 주는 역할을 한다. CNN에서는 컨볼루션 레이어나 맥스풀링 레이어를 반복적으로 거치면 주요 특징만 추출되고, 추출된 주요 특징은 전결합층에 전달되어 학습된다. 컨볼루션 레이어나 맥스풀링 레이어는 주로 2차원 자료를 다루지만 전결합층에 전달하기 위해선 1차원 자료로 바꿔 줘야 한다. 이때 사용되는 것이 플래튼 레이어다. 이전 레이어의 출력 정보를 이용하여 입력 정보를 자동으로 설정되며, 출력 형태는 입력 형태에 따라 자동으로 계산되기 때문에 별도로 사용자가 파라미터를 지정해 주지 않아도 된다.
* 덴스 (Dense) 레이어는 모든 입력 뉴런과 출력 뉴런을 연결하는 전결합층이다.
* 드롭아웃 (Dropout) 레이어는 신경망에서 가장 효과적이고 널리 사용하는 규제 기법 중 하나이다. 드롭아웃 레이어에 적용하면 훈련하는 동안 레이어의 출력 특성을 랜덤하게 0으로 만든다. 보통 0.2 에서 0.5 사이를 사용하고 (여기서는 0.02, 0.3, 0.4를 이용하였다.), 이를 통해 Overfitting을 감소시킨다.
* 배치 정규화 (Batch Normalization) 레이어는 덴스 레이어 값들을 활성화 함수로 넘겨 주기 전에 정규화를 한 후 비선형 활성화 함수를 적용하는 역할을 한다. 이를 통해 신경망의 각 레이어의 분포를 같게 함으로써 좀 더 안정적인 학습이 가능하다.
* 이러한 레이어들을 쌓고 쌓아 정확도를 어느 정도 높일 수 있었다. 이 모델을 가시화하면 아래 그림과 같이 나타난다.



**① ②**



**③ ④**

* **Use other optimizer**
* 앞서 구현한 모델과 잘 맞는 optimizer(최적화기)을 찾기 위해 Adagrad, RMSProp, AdaDelta, Adam, Adamax 등을 써 본 결과 Adamax가 가장 높은 정확도를 보였다. Adamax는 무한 norm을 기반으로 한 Adam의 변형 버전이다. 기본 매개변수 값들은 논문에 제시된 값을 따른다. 인자들은 다음과 같다.

ⓐ lr : float >= 0. 학습율을 뜻한다. 여기서는 0.002로 설정하였다.

ⓑ beta\_1/beta\_2 : floats, 0 < beta <1. 일반적으로 1에 가깝다.

ⓒ epsilon : float >= 0. 퍼징 인자이다.

ⓓ decay : float >= 0. 각 갱신마다의 학습율 감쇄를 뜻한다.

* **Adjust the learning rate (Between 0 and 1)**
* optimizer 부분에서 learning rate를 뜻하는 변수인 lr을 0.002에서 0.0009로 낮춘 결과 근소한 차이지만 조금 더 높은 정확도를 보였다.

위 방법들을 통해 정확도가 이전보다 확실히 높아진 것을 알 수 있다. 실행해 보고 나서 정확도는 약 85% 정도로 나타났다. 한 epoch가 실행될수록 loss는 점점 낮아졌고, 정확도는 점점 높아지는 경향을 보였다.

1. **코드**

|  |
| --- |
| **20171672.ipynb** |
| **from \_\_future\_\_ import print\_function**  **import keras**  **from keras.datasets import cifar10**  **from keras.preprocessing.image import ImageDataGenerator**  **from keras.models import Sequential**  **from keras.layers import Dense, Dropout, Activation, Flatten, BatchNormalization**  **from keras.layers import Conv2D, MaxPooling2D**  **from keras import regularizers, optimizers**  **from keras.utils.vis\_utils import model\_to\_dot**  **%matplotlib inline**  **import matplotlib.pyplot as plt**  **from sklearn.metrics import confusion\_matrix**  **import pandas as pd**  **import seaborn as sns**  **import os**  **# Import modules**  **batch\_size = 32**  **num\_classes = 10**  **epochs = 40**  **data\_augmentation = False**  **num\_predictions = 20**  **save\_dir = os.path.join(os.getcwd(), 'saved\_models')**  **model\_name = 'keras\_cifar10\_trained\_model.h5'**  **# Tune parameters**  **def plot\_images(x, y\_true, y\_pred=None, size=(5, 5)):**  **assert len(x) == len(y\_true) == size[0] \* size[1]**    **fig, axes = plt.subplots(size[0], size[1])**  **fig.subplots\_adjust(hspace=0.5, wspace=0.1)**  **for i, ax in enumerate(axes.flat):**  **if x[i].shape[-1] == 1:**  **ax.imshow(x[i].reshape(x[i].shape[0], x[i].shape[1]))**  **else:**  **ax.imshow(x[i])**  **if y\_pred is None:**  **xlabel = "True: {0}".format(y\_true[i].argmax())**  **else:**  **xlabel = "True: {0}, Pred: {1}".format(y\_true[i].argmax(),**  **y\_pred[i].argmax())**  **ax.set\_xlabel(xlabel)**    **ax.set\_xticks([])**  **ax.set\_yticks([])**  **plt.show()**    **(x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data()**  **print('x\_train shape:', x\_train.shape)**  **print(x\_train.shape[0], 'train samples')**  **print(x\_test.shape[0], 'test samples')**  **y\_train = keras.utils.to\_categorical(y\_train, num\_classes)**  **y\_test = keras.utils.to\_categorical(y\_test, num\_classes)**  **# Load data**  **plot\_images(x\_train[:25], y\_train[:25])**  **model = Sequential()**  **weight\_decay = 1e-4**  **model.add(Conv2D(32, (3,3), padding='same', kernel\_regularizer=regularizers.l2(weight\_decay), input\_shape=x\_train.shape[1:]))**  **model.add(Activation('relu'))**  **model.add(BatchNormalization())**  **model.add(Conv2D(32, (3, 3), padding='same', kernel\_regularizer=regularizers.l2(weight\_decay)))**  **model.add(Activation('relu'))**  **model.add(BatchNormalization())**  **model.add(MaxPooling2D(pool\_size=(2, 2)))**  **model.add(Dropout(0.2))**  **model.add(Conv2D(64, (3, 3), padding='same', kernel\_regularizer=regularizers.l2(weight\_decay)))**  **model.add(Activation('relu'))**  **model.add(BatchNormalization())**  **model.add(Conv2D(64, (3, 3), padding='same', kernel\_regularizer=regularizers.l2(weight\_decay)))**  **model.add(Activation('relu'))**  **model.add(BatchNormalization())**  **model.add(MaxPooling2D(pool\_size=(2, 2)))**  **model.add(Dropout(0.3))**  **model.add(Conv2D(4\*32, (3,3), padding='same', kernel\_regularizer=regularizers.l2(weight\_decay)))**  **model.add(Activation('relu'))**  **model.add(BatchNormalization())**  **model.add(Conv2D(4\*32, (3,3), padding='same', kernel\_regularizer=regularizers.l2(weight\_decay)))**  **model.add(Activation('relu'))**  **model.add(BatchNormalization())**  **model.add(MaxPooling2D(pool\_size=(2,2)))**  **model.add(Dropout(0.4))**  **model.add(Flatten())**  **model.add(Dense(num\_classes))**  **model.add(Activation('softmax'))**  **# Create model**  **opt = keras.optimizers.Adamax(lr=0.0009, beta\_1=0.9, beta\_2=0.999, epsilon=None, decay=0.0) # initiate Adamax optimizer**  **# Set up learning course of the model**  **model.compile(loss='categorical\_crossentropy',**  **optimizer=opt,**  **metrics=['accuracy'])**  **x\_train = x\_train.astype('float32')**  **x\_test = x\_test.astype('float32')**  **x\_train /= 255**  **x\_test /= 255**  **if not data\_augmentation:**  **print('Not using data augmentation.')**  **model.fit(x\_train, y\_train,**  **batch\_size=batch\_size,**  **epochs=epochs,**  **validation\_data=(x\_test, y\_test),**  **shuffle=True)**  **else:**  **print('Using real-time data augmentation.')**  **# This will do preprocessing and realtime data augmentation:**  **datagen = ImageDataGenerator(**  **featurewise\_center=False, # set input mean to 0 over the dataset**  **samplewise\_center=False, # set each sample mean to 0**  **featurewise\_std\_normalization=False, # divide inputs by std of the dataset**  **samplewise\_std\_normalization=False, # divide each input by its std**  **zca\_whitening=False, # apply ZCA whitening**  **zca\_epsilon=1e-06, # epsilon for ZCA whitening**  **rotation\_range=0, # randomly rotate images in the range (degrees, 0 to 180)**  **# randomly shift images horizontally (fraction of total width)**  **width\_shift\_range=0.1,**  **# randomly shift images vertically (fraction of total height)**  **height\_shift\_range=0.1,**  **shear\_range=0., # set range for random shear**  **zoom\_range=0., # set range for random zoom**  **channel\_shift\_range=0., # set range for random channel shifts**  **# set mode for filling points outside the input boundaries**  **fill\_mode='nearest',**  **cval=0., # value used for fill\_mode = "constant"**  **horizontal\_flip=True, # randomly flip images**  **vertical\_flip=False, # randomly flip images**  **# set rescaling factor (applied before any other transformation)**  **rescale=None,**  **# set function that will be applied on each input**  **preprocessing\_function=None,**  **# image data format, either "channels\_first" or "channels\_last"**  **data\_format=None,**  **# fraction of images reserved for validation (strictly between 0 and 1)**  **validation\_split=0.0)**  **# Compute quantities required for feature-wise normalization**  **# (std, mean, and principal components if ZCA whitening is applied).**  **datagen.fit(x\_train)**  **# Fit the model on the batches generated by datagen.flow().**  **model.fit\_generator(datagen.flow(x\_train, y\_train,**  **batch\_size=batch\_size),**  **steps\_per\_epoch = 100,**  **epochs=epochs,**  **validation\_data=(x\_test, y\_test),**  **workers=4)**  **# Compile and train the model**  **if not os.path.isdir(save\_dir):**  **os.makedirs(save\_dir)**  **model\_path = os.path.join(save\_dir, model\_name)**  **model.save(model\_path)**  **print('Saved trained model at %s ' % model\_path)**  **scores = model.evaluate(x\_test, y\_test, verbose=1)**  **#Evaluate the model**  **print('Test loss:', scores[0])**  **print('Test accuracy:', scores[1]\*100, '%')**  **# Print the loss and accuracy**  **y\_pred = model.predict(x\_test)**  **plot\_images(x=x\_test[:25], y\_true=y\_test[:25], y\_pred=y\_pred[:25])**  **y\_result = confusion\_matrix(y\_test.argmax(axis=1), y\_pred.argmax(axis=1))**  **sns.heatmap(pd.DataFrame(y\_result, range(10), range(10)), annot=True, fmt='g')** |