# 使用Python、AI与Kubernetes构建可扩展图像分类服务

## 课程目标

1. 学生能够理解如何使用Python和深度学习框架（如TensorFlow）构建基本的图像分类AI模型。
2. 掌握在PyCharm中开发和调试Python代码的技巧。
3. 理解Kubernetes（K8s）的基本概念，并学会使用K8s对AI服务进行容器化部署和管理，实现服务的可扩展性。

## 课程安排

### 第一节课：Python与AI模型构建

1. **课程导入（10分钟）**
   * 介绍图像分类在现实生活中的应用场景，如医疗影像诊断、自动驾驶中的目标识别等。
   * 展示Kubernetes在大规模应用部署和管理中的重要性，例如如何实现高可用和弹性扩展。
2. **环境搭建（20分钟）**
   * 确保学生安装好PyCharm、Python 3.x以及相关依赖包（如TensorFlow、Keras、OpenCV）。
   * 在PyCharm中创建一个新的Python项目。
3. **图像分类AI模型开发（40分钟）**
   * 讲解深度学习图像分类的基本原理，如卷积神经网络（CNN）的结构和工作机制。
   * 演示如何使用Keras和TensorFlow构建一个简单的CNN图像分类模型，以MNIST数据集为例：

|  |
| --- |
| import tensorflow as tf  from tensorflow.keras.datasets import mnist  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense  # 加载MNIST数据集  (train\_images, train\_labels), (test\_images, test\_labels) = mnist.load\_data()  # 数据预处理  train\_images = train\_images.reshape((-1, 28, 28, 1)).astype('float32') / 255.0  test\_images = test\_images.reshape((-1, 28, 28, 1)).astype('float32') / 255.0  # 构建模型  model = Sequential([  Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)),  MaxPooling2D((2, 2)),  Conv2D(64, (3, 3), activation='relu'),  MaxPooling2D((2, 2)),  Flatten(),  Dense(64, activation='relu'),  Dense(10, activation='softmax')  ])  # 编译模型  model.compile(optimizer='adam',  loss='sparse\_categorical\_crossentropy',  metrics=['accuracy'])  # 训练模型  model.fit(train\_images, train\_labels, epochs=5, batch\_size=64)  # 评估模型  test\_loss, test\_acc = model.evaluate(test\_images, test\_labels)  print(f"Test accuracy: {test\_acc}") |

* 在PyCharm中逐步运行代码，讲解每一步的作用，包括数据加载、预处理、模型构建、编译、训练和评估。

1. **模型保存（10分钟）**
   * 讲解如何将训练好的模型保存下来，以便后续部署使用：

|  |
| --- |
| model.save('mnist\_model.h5') |

### 第二节课：Kubernetes部署与管理

1. **Kubernetes基础概念讲解（20分钟）**
   * 介绍Kubernetes的核心概念，如Pod、Service、Deployment等。
   * 解释容器化的概念，以及为什么在K8s中容器是部署和管理应用的基本单元。
2. **容器化AI服务（30分钟）**
   * 讲解如何使用Docker将训练好的图像分类模型及其依赖打包成容器镜像。首先，创建一个 Dockerfile：

|  |
| --- |
| FROM python:3.8  WORKDIR /app  COPY requirements.txt.  RUN pip install -r requirements.txt  COPY. /app  CMD ["python", "app.py"] |

* 在 requirements.txt 文件中列出项目所需的依赖包：

|  |
| --- |
| tensorflow  numpy |

* 编写 app.py 文件，用于加载模型并提供简单的图像分类预测接口：

|  |
| --- |
| import tensorflow as tf  import numpy as np  from flask import Flask, request, jsonify  app = Flask(\_\_name\_\_)  model = tf.keras.models.load\_model('mnist\_model.h5')  @app.route('/predict', method=['POST'])  def predict():  data = request.get\_json(force=True)  image = np.array(data['image']).reshape((1, 28, 28, 1)).astype('float32') / 255.0  prediction = model.predict(image).tolist()  return jsonify({'prediction': prediction})  if \_\_name\_\_ == '\_\_main\_\_':  app.run(host='0.0.0.0', port=5000) |

* 使用命令 docker build -t mnist - classifier:v1. 构建容器镜像。

1. **Kubernetes部署（30分钟）**
   * 确保学生已安装并配置好Kubernetes环境（如Minikube）。
   * 创建一个Deployment配置文件 mnist - deployment.yaml：

|  |
| --- |
| apiVersion: apps/v1  kind: Deployment  metadata:  name: mnist - classifier - deployment  spec:  replicas: 3  selector:  matchLabels:  app: mnist - classifier  template:  metadata:  labels:  app: mnist - classifier  spec:  containers:  - name: mnist - classifier  image: mnist - classifier:v1  ports:  - containerPort: 5000 |

* 创建一个Service配置文件 mnist - service.yaml：

|  |
| --- |
| apiVersion: v1  kind: Service  metadata:  name: mnist - classifier - service  spec:  selector:  app: mnist - classifier  ports:  - protocol: TCP  port: 80  targetPort: 5000  type: LoadBalancer |

* 使用命令 kubectl apply - f mnist - deployment.yaml 和 kubectl apply - f mnist - service.yaml 部署应用。
* 通过命令 kubectl get pods 和 kubectl get services 查看部署状态和服务地址，然后使用浏览器或工具调用服务接口进行图像分类预测。

1. **课程总结与拓展（10分钟）**
   * 回顾从Python开发AI模型到使用Kubernetes进行容器化部署和管理的整个流程。
   * 提出拓展方向，如使用Kubernetes的自动伸缩功能根据负载动态调整Pod数量，或使用更复杂的图像数据集和模型结构进行图像分类。