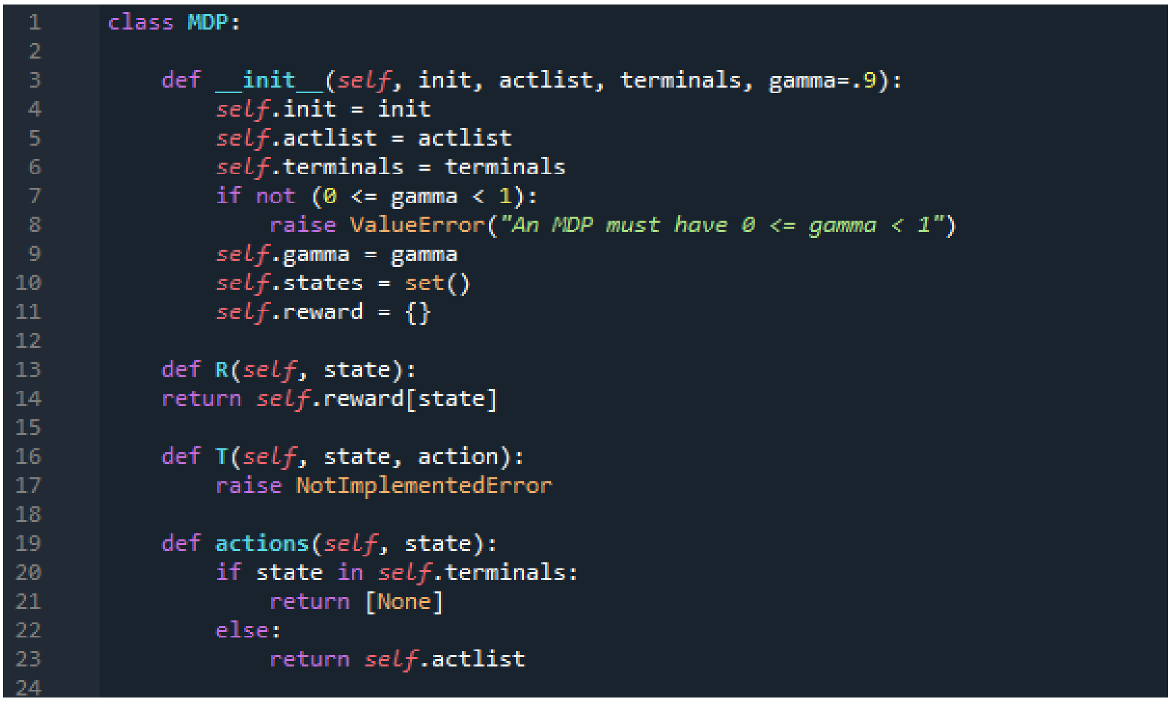
目　次

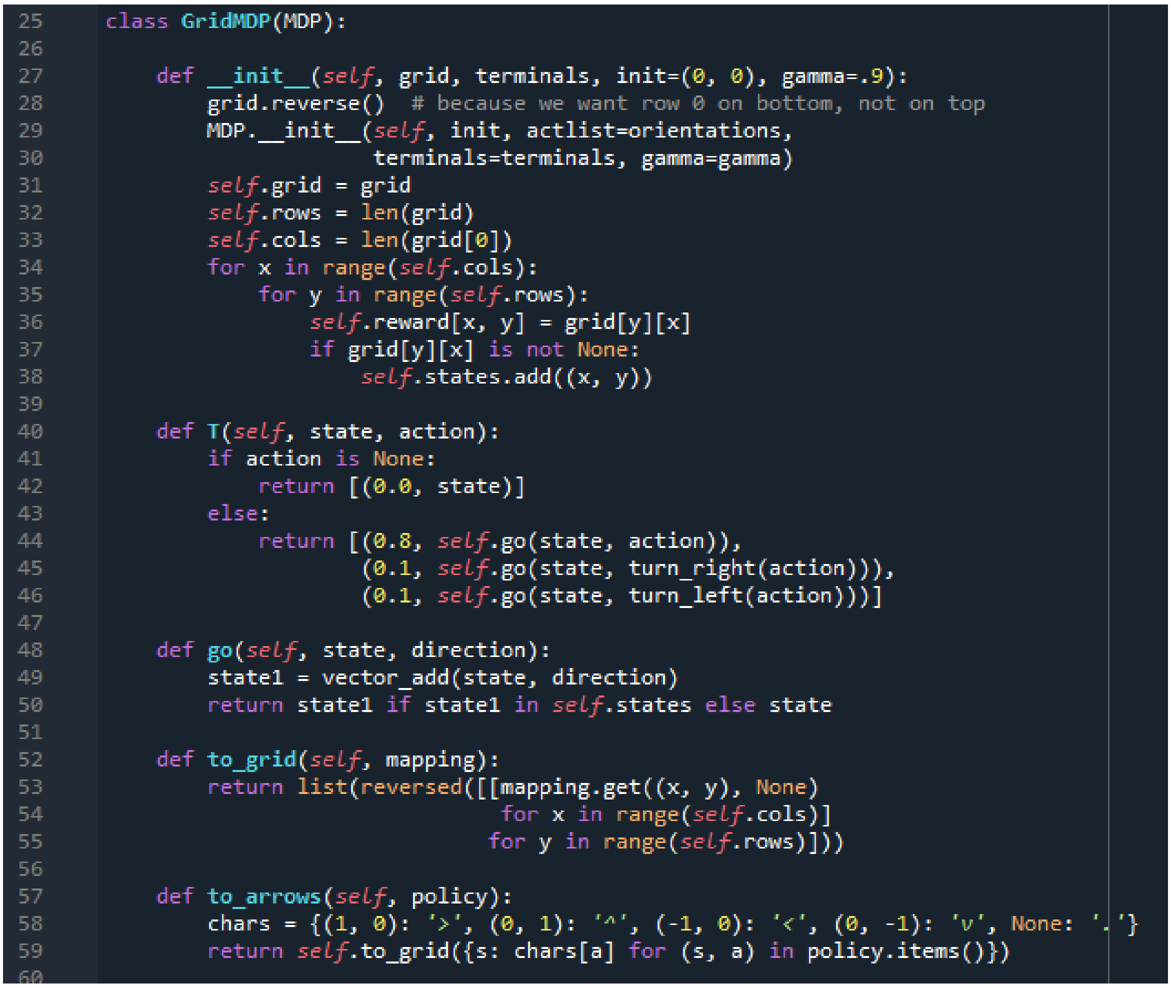
内容

１． section1_強化学習

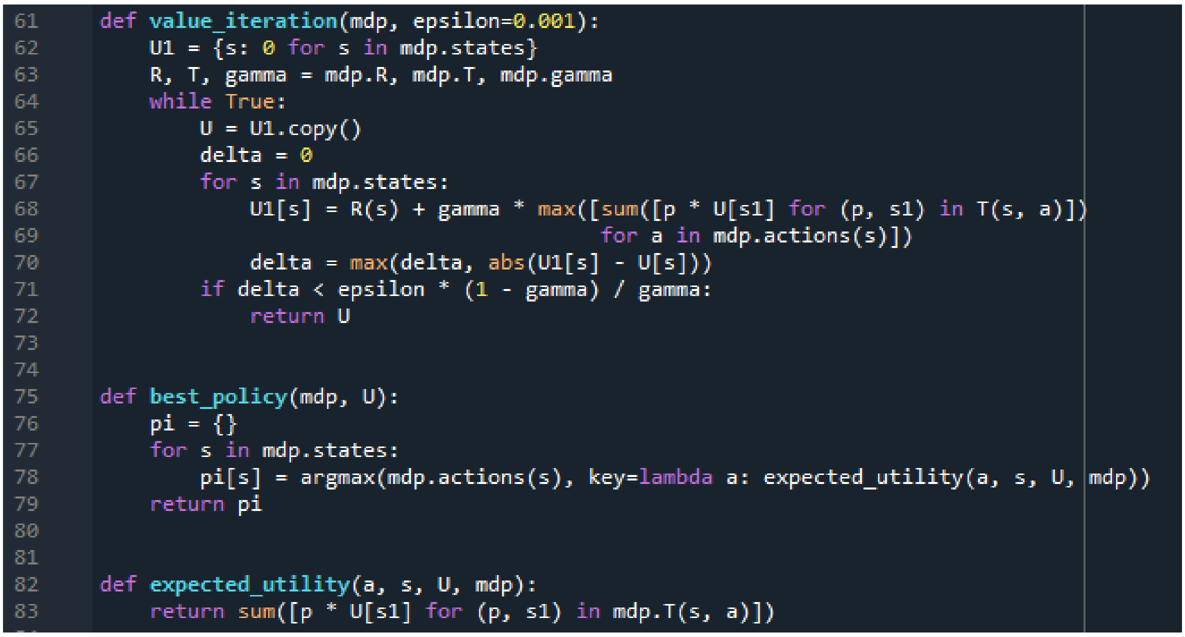
マルコフ決定過程（Markov decision process：MDP）



```python
class MDP:

    def __init__(self, init, actlist, terminals, gamma=.9):
        self.init = init
        self.actlist = actlist
        self.terminals = terminals
        if not (0 <= gamma < 1):
            raise ValueError("An MDP must have 0 <= gamma < 1")
        self.gamma = gamma
        self.states = set()
        self.reward = {}

    def R(self, state):
    return self.reward[state]

    def T(self, state, action):
        raise NotImplementedError

    def actions(self, state):
        if state in self.terminals:
            return [None]
        else:
            return self.actlist

```



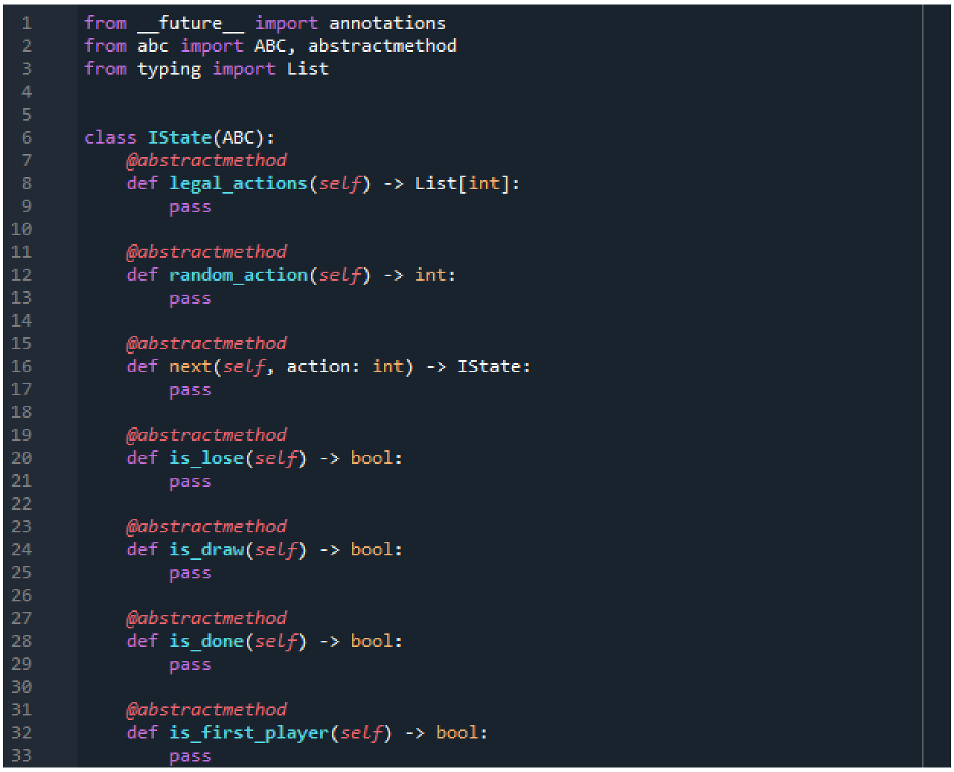
```python
class GridMDP(MDP):

    def __init__(self, grid, terminals, init=(0, 0), gamma=.9):
        grid.reverse()  # because we want row 0 on bottom, not on top
        MDP.__init__(self, init, actlist=orientations,
                     terminals=terminals, gamma=gamma)
        self.grid = grid
        self.rows = len(grid)
        self.cols = len(grid[0])
        for x in range(self.cols):
            for y in range(self.rows):
                self.reward[x, y] = grid[y][x]
                if grid[y][x] is not None:
                    self.states.add((x, y))

    def T(self, state, action):
        if action is None:
            return [(0.0, state)]
        else:
            return [(0.8, self.go(state, action)),
                    (0.1, self.go(state, turn_right(action))),
                    (0.1, self.go(state, turn_left(action)))]

    def go(self, state, direction):
        state1 = vector_add(state, direction)
        return state1 if state1 in self.states else state

    def to_grid(self, mapping):
        return list(reversed([[mapping.get((x, y), None)
                               for x in range(self.cols)]
                              for y in range(self.rows)]))

    def to_arrows(self, policy):
        chars = {(1, 0): '>', (0, 1): '^', (-1, 0): '<', (0, -1): 'v', None: '.'}
        return self.to_grid({s: chars[a] for (s, a) in policy.items()})

```



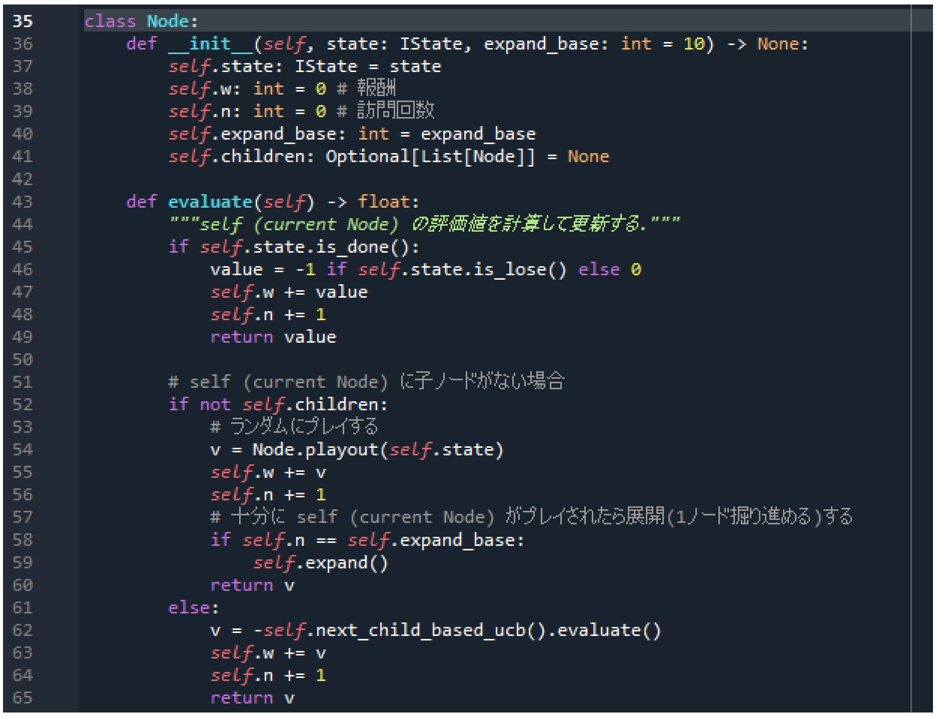
```python
def value_iteration(mdp, epsilon=0.001):
    U1 = {s: 0 for s in mdp.states}
    R, T, gamma = mdp.R, mdp.T, mdp.gamma
    while True:
        U = U1.copy()
        delta = 0
        for s in mdp.states:
            U1[s] = R(s) + gamma * max([sum([p * U[s1] for (p, s1) in T(s, a)])
                                        for a in mdp.actions(s)])
            delta = max(delta, abs(U1[s] - U[s]))
        if delta < epsilon * (1 - gamma) / gamma:
            return U


def best_policy(mdp, U):
    pi = {}
    for s in mdp.states:
        pi[s] = argmax(mdp.actions(s), key=lambda a: expected_utility(a, s, U, mdp))
    return pi


def expected_utility(a, s, U, mdp):
    return sum([p * U[s1] for (p, s1) in mdp.T(s, a)])
```

２．　section2_AlphaGo

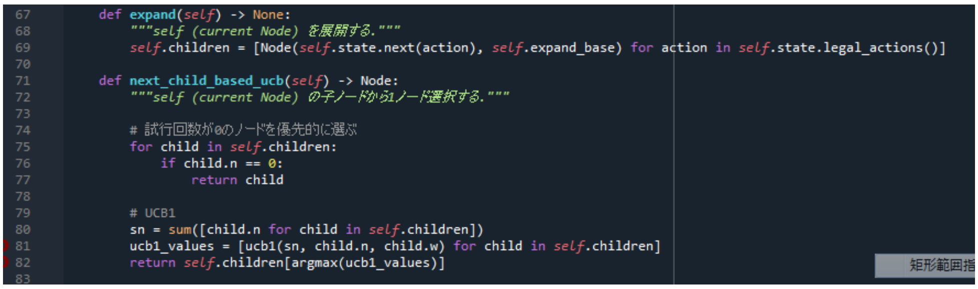
AlphaGo Zero　モンテカルロ木探索の実装



```python
from __future__ import annotations
from abc import ABC, abstractmethod
from typing import List


class IState(ABC):
    @abstractmethod
    def legal_actions(self) -> List[int]:
        pass

    @abstractmethod
    def random_action(self) -> int:
        pass

    @abstractmethod
    def next(self, action: int) -> IState:
        pass

    @abstractmethod
    def is_lose(self) -> bool:
        pass

    @abstractmethod
    def is_draw(self) -> bool:
        pass

    @abstractmethod
    def is_done(self) -> bool:
        pass

    @abstractmethod
    def is_first_player(self) -> bool:
        pass
```



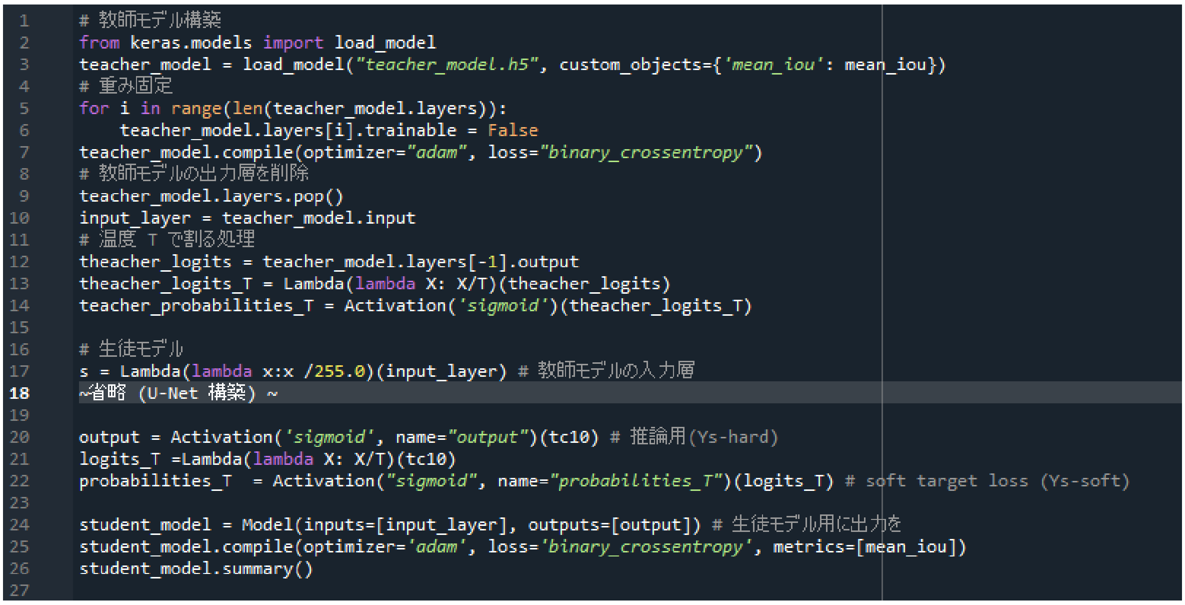
```python
class Node:
    def __init__(self, state: IState, expand_base: int = 10) -> None:
        self.state: IState = state
        self.w: int = 0 # 報酬
        self.n: int = 0 # 訪問回数
        self.expand_base: int = expand_base
        self.children: Optional[List[Node]] = None

    def evaluate(self) -> float:
        """self (current Node) の評価値を計算して更新する."""
        if self.state.is_done():
            value = -1 if self.state.is_lose() else 0
            self.w += value
            self.n += 1
            return value

        # self (current Node) に子ノードがない場合
        if not self.children:
            # ランダムにプレイする
            v = Node.playout(self.state)
            self.w += v
            self.n += 1
            # 十分に self (current Node) がプレイされたら展開(1ノード掘り進める)する
            if self.n == self.expand_base:
                self.expand()
            return v
        else:
            v = -self.next_child_based_ucb().evaluate()
            self.w += v
            self.n += 1
            return v
```



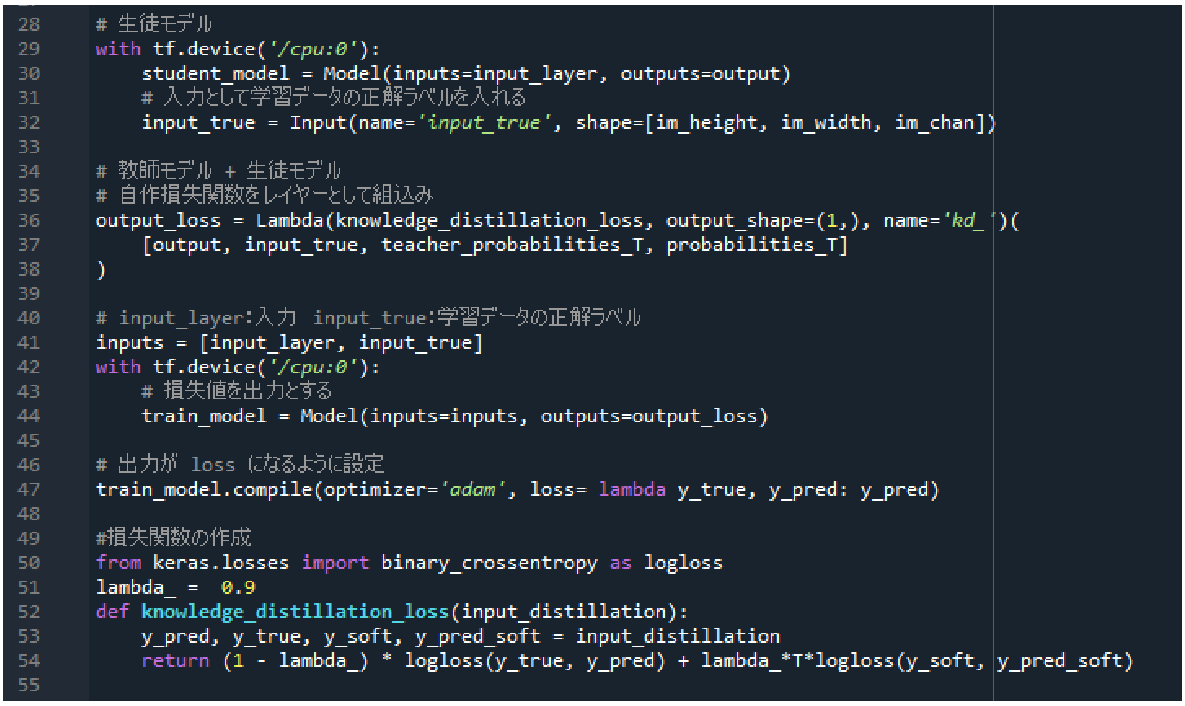
```python
    def expand(self) -> None:
        """self (current Node) を展開する."""
        self.children = [Node(self.state.next(action), self.expand_base) for action in self.state.legal_actions()]

    def next_child_based_ucb(self) -> Node:
        """self (current Node) の子ノードから1ノード選択する."""

        # 試行回数が0のノードを優先的に選ぶ
        for child in self.children:
            if child.n == 0:
                return child

        # UCB1
        sn = sum([child.n for child in self.children])
        ucb1_values = [ucb1(sn, child.n, child.w) for child in self.children]
        return self.children[argmax(ucb1_values)]
```

３．　section3_軽量化_高速化技術

蒸留実装



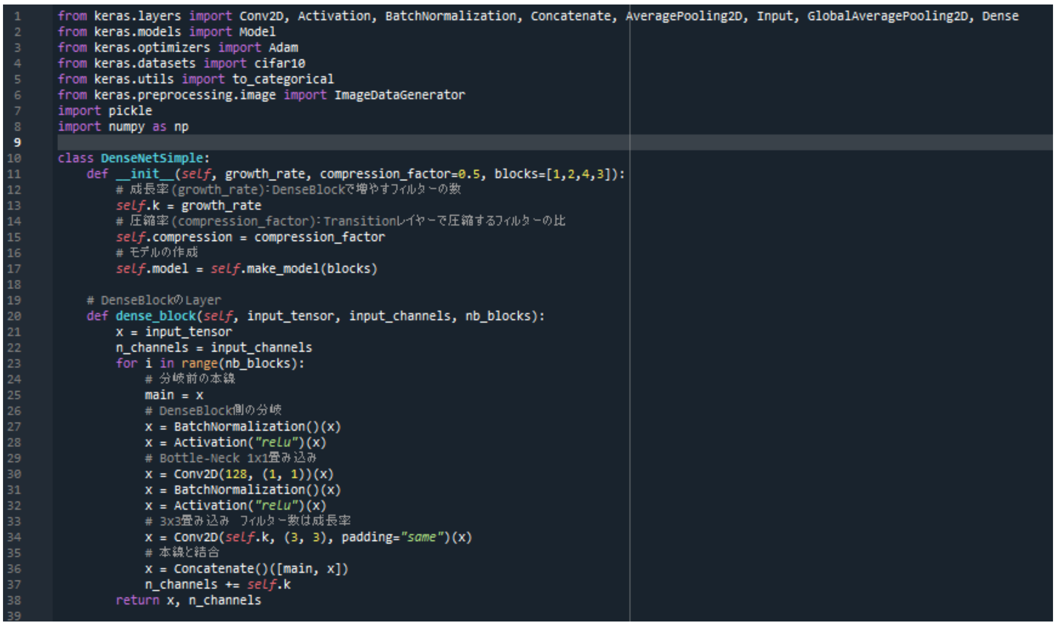
```
# 教師モデル構築
from keras.models import load_model
teacher_model = load_model("teacher_model.h5", custom_objects={'mean_iou': mean_iou})
# 重み固定
for i in range(len(teacher_model.layers)):
    teacher_model.layers[i].trainable = False
teacher_model.compile(optimizer="adam", loss="binary_crossentropy")
# 教師モデルの出力層を削除
teacher_model.layers.pop()
input_layer = teacher_model.input
# 温度 T で割る処理
theacher_logits = teacher_model.layers[-1].output
theacher_logits_T = Lambda(lambda X: X/T)(theacher_logits)
teacher_probabilities_T = Activation('sigmoid')(theacher_logits_T)

# 生徒モデル
s = Lambda(lambda x:x /255.0)(input_layer) # 教師モデルの入力層
~省略 (U-Net 構築) ~

output = Activation('sigmoid', name="output")(tc10) # 推論用(Ys-hard)
logits_T =Lambda(lambda X: X/T)(tc10)
probabilities_T  = Activation("sigmoid", name="probabilities_T")(logits_T) # soft target loss (Ys-soft)

student_model = Model(inputs=[input_layer], outputs=[output]) # 生徒モデル用に出力を
student_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=[mean_iou])
student_model.summary()

```



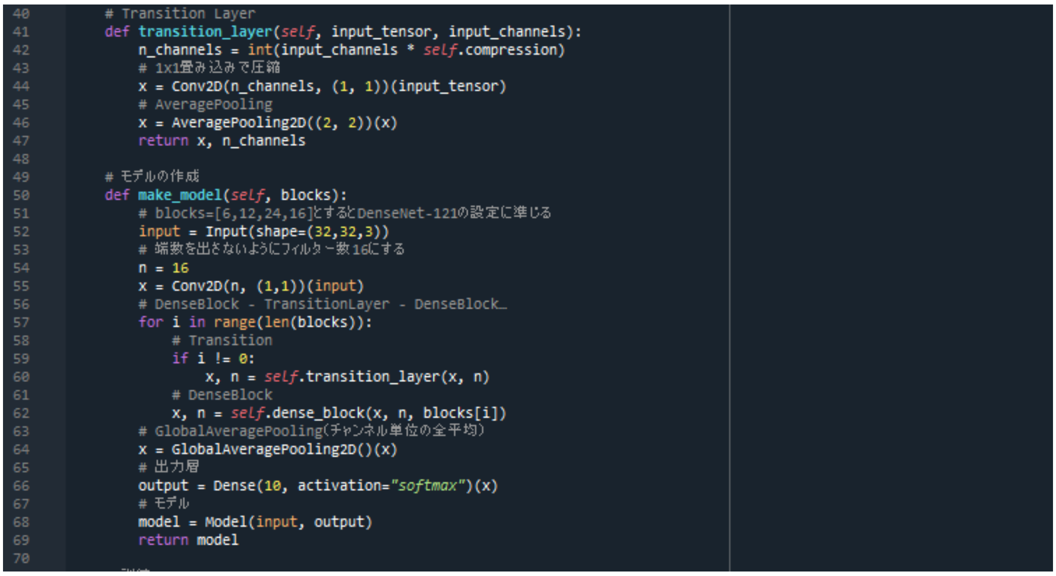
```
# 生徒モデル
with tf.device('/cpu:0'):
    student_model = Model(inputs=input_layer, outputs=output)
    # 入力として学習データの正解ラベルを入れる
    input_true = Input(name='input_true', shape=[im_height, im_width, im_chan])

# 教師モデル + 生徒モデル
# 自作損失関数をレイヤーとして組込み
output_loss = Lambda(knowledge_distillation_loss, output_shape=(1,), name='kd_')(
    [output, input_true, teacher_probabilities_T, probabilities_T]
)

# input_layer:入力 input_true:学習データの正解ラベル
inputs = [input_layer, input_true]
with tf.device('/cpu:0'):
    # 損失値を出力とする
    train_model = Model(inputs=inputs, outputs=output_loss)

# 出力が loss になるように設定
train_model.compile(optimizer='adam', loss= lambda y_true, y_pred: y_pred)

#損失関数の作成
from keras.losses import binary_crossentropy as logloss
lambda_ =  0.9
def knowledge_distillation_loss(input_distillation):
    y_pred, y_true, y_soft, y_pred_soft = input_distillation
    return (1 - lambda_) * logloss(y_true, y_pred) + lambda_*T*logloss(y_soft, y_pred_soft)

```

４．　section4_応用モデル

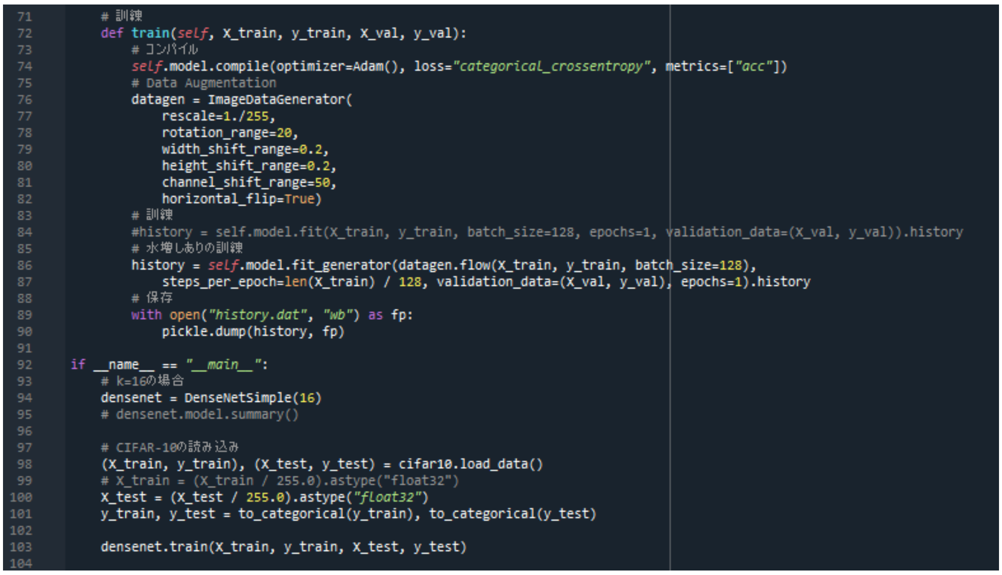
DenseNet　実装



```python
from keras.layers import Conv2D, Activation, BatchNormalization, Concatenate, AveragePooling2D, Input, GlobalAveragePooling2D, Dense
from keras.models import Model
from keras.optimizers import Adam
from keras.datasets import cifar10
from keras.utils import to_categorical
from keras.preprocessing.image import ImageDataGenerator
import pickle
import numpy as np

class DenseNetSimple:
    def __init__(self, growth_rate, compression_factor=0.5, blocks=[1,2,4,3]):
        # 成長率(growth_rate):DenseBlockで増やすフィルターの数
        self.k = growth_rate
        # 圧縮率(compression_factor):Transitionレイヤーで圧縮するフィルターの比
        self.compression = compression_factor
        # モデルの作成
        self.model = self.make_model(blocks)

    # DenseBlockのLayer
    def dense_block(self, input_tensor, input_channels, nb_blocks):
        x = input_tensor
        n_channels = input_channels
        for i in range(nb_blocks):
            # 分岐前の本線
            main = x
            # DenseBlock側の分岐
            x = BatchNormalization()(x)
            x = Activation("relu")(x)
            # Bottle-Neck 1x1畳み込み
            x = Conv2D(128, (1, 1))(x)
            x = BatchNormalization()(x)
            x = Activation("relu")(x)
            # 3x3畳み込み フィルター数は成長率
            x = Conv2D(self.k, (3, 3), padding="same")(x)
            # 本線と結合
            x = Concatenate()([main, x])
            n_channels += self.k
        return x, n_channels

```



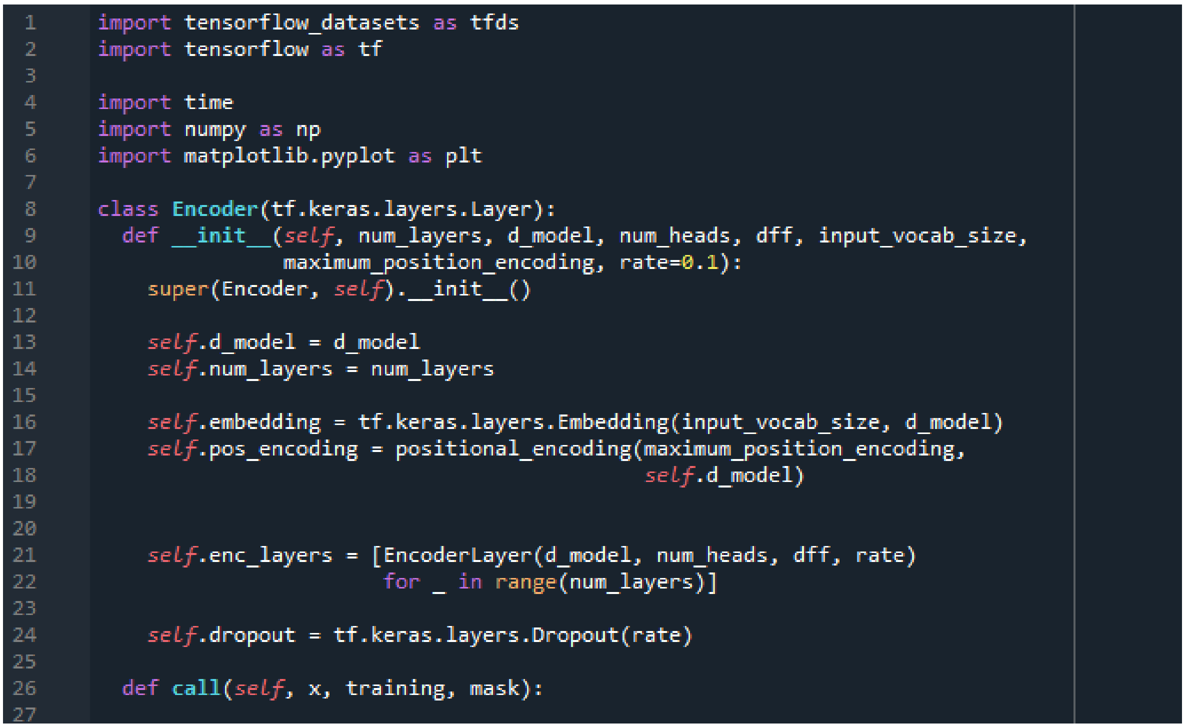
```python
    # Transition Layer
    def transition_layer(self, input_tensor, input_channels):
        n_channels = int(input_channels * self.compression)
        # 1x1畳み込みで圧縮
        x = Conv2D(n_channels, (1, 1))(input_tensor)
        # AveragePooling
        x = AveragePooling2D((2, 2))(x)
        return x, n_channels

    # モデルの作成
    def make_model(self, blocks):
        # blocks=[6,12,24,16]とするとDenseNet-121の設定に準じる
        input = Input(shape=(32,32,3))
        # 端数を出さないようにフィルター数16にする
        n = 16
        x = Conv2D(n, (1,1))(input)
        # DenseBlock - TransitionLayer - DenseBlock…
        for i in range(len(blocks)):
            # Transition
            if i != 0:
                x, n = self.transition_layer(x, n)
            # DenseBlock
            x, n = self.dense_block(x, n, blocks[i])
        # GlobalAveragePooling(チャンネル単位の全平均)
        x = GlobalAveragePooling2D()(x)
        # 出力層
        output = Dense(10, activation="softmax")(x)
        # モデル
        model = Model(input, output)
        return model

```



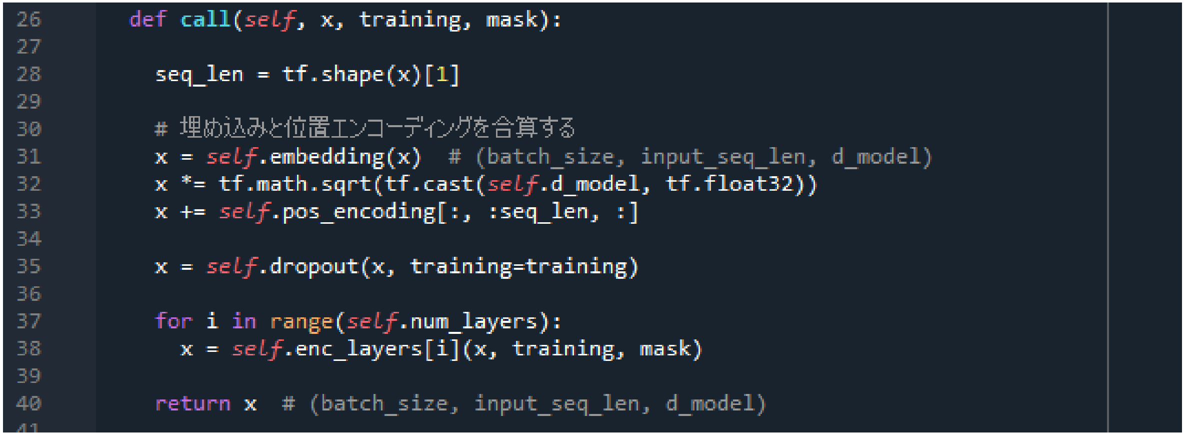
```python
    # 訓練
    def train(self, X_train, y_train, X_val, y_val):
        # コンパイル
        self.model.compile(optimizer=Adam(), loss="categorical_crossentropy", metrics=["acc"])
        # Data Augmentation
        datagen = ImageDataGenerator(
            rescale=1./255,
            rotation_range=20,
            width_shift_range=0.2,
            height_shift_range=0.2,
            channel_shift_range=50,
            horizontal_flip=True)
        # 訓練
        #history = self.model.fit(X_train, y_train, batch_size=128, epochs=1, validation_data=(X_val, y_val)).history
        # 水増しありの訓練
        history = self.model.fit_generator(datagen.flow(X_train, y_train, batch_size=128),
            steps_per_epoch=len(X_train) / 128, validation_data=(X_val, y_val), epochs=1).history
        # 保存
        with open("history.dat", "wb") as fp:
            pickle.dump(history, fp)

if __name__ == "__main__":
    # k=16の場合
    densenet = DenseNetSimple(16)
    # densenet.model.summary()

    # CIFAR-10の読み込み
    (X_train, y_train), (X_test, y_test) = cifar10.load_data()
    # X_train = (X_train / 255.0).astype("float32")
    X_test = (X_test / 255.0).astype("float32")
    y_train, y_test = to_categorical(y_train), to_categorical(y_test)

    densenet.train(X_train, y_train, X_test, y_test)

```

5． section5_Transformer

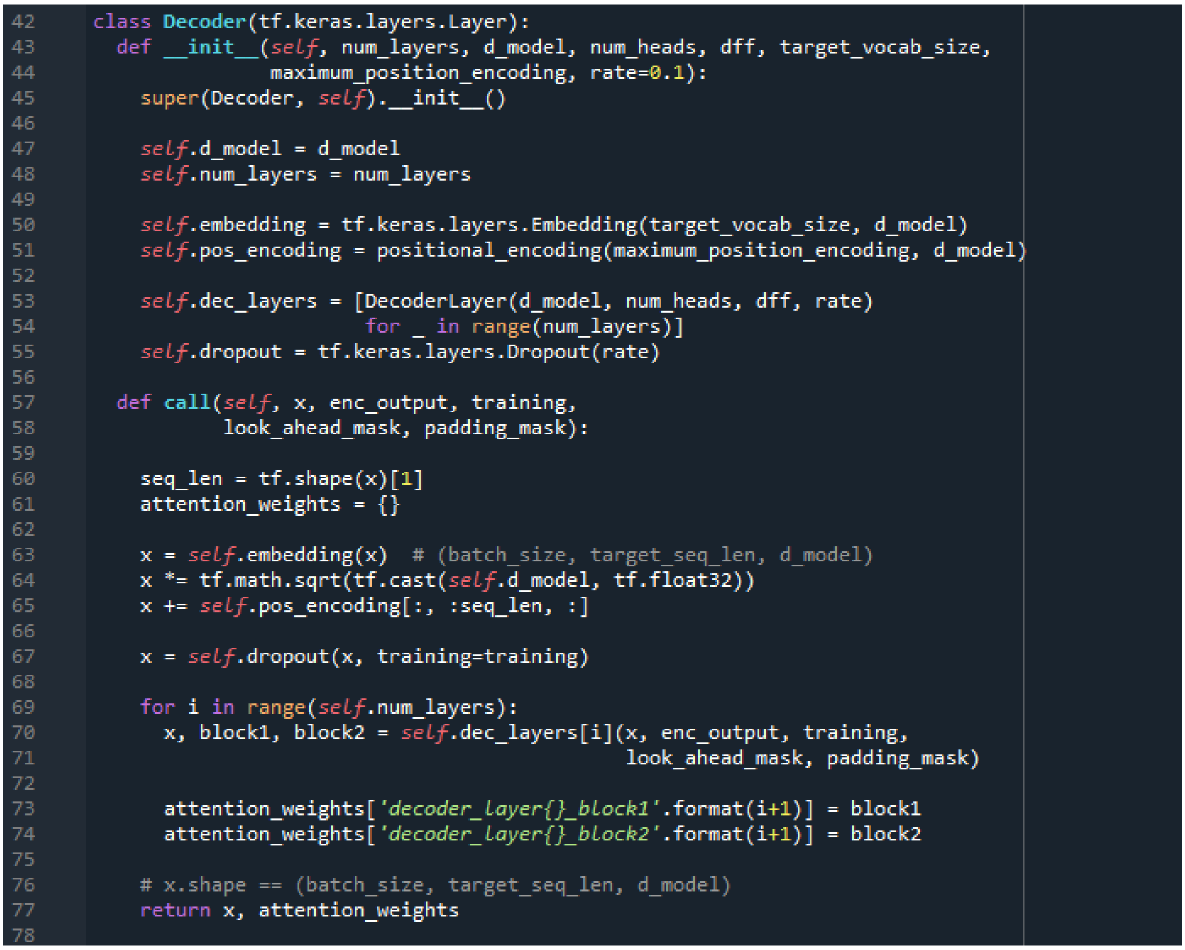
Transformer,Encoder,Decoder



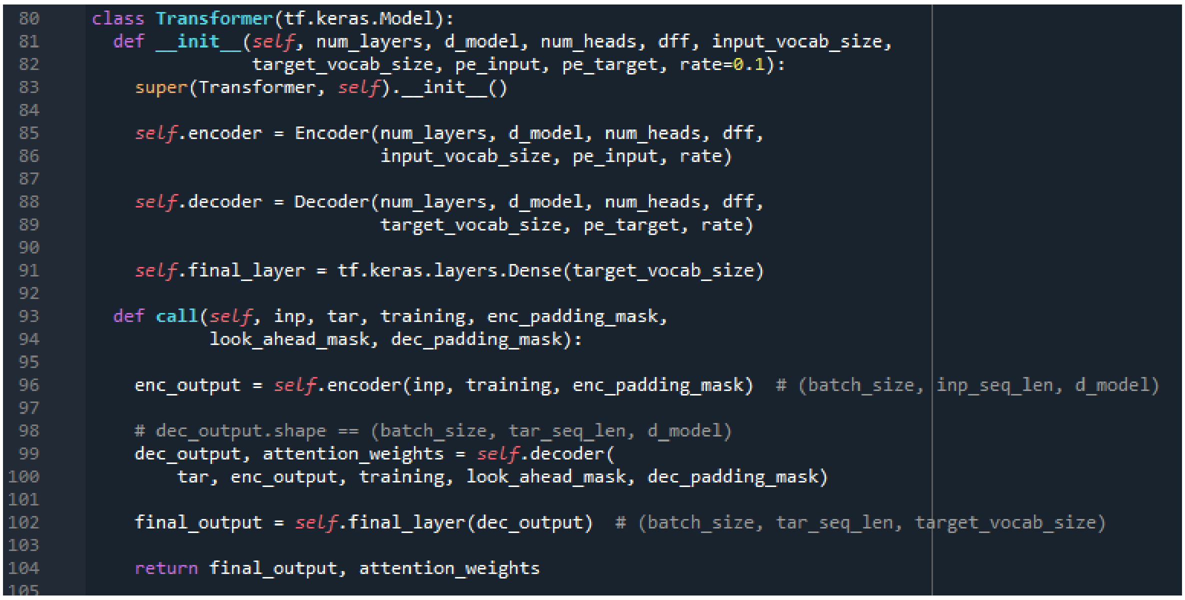
```python
import tensorflow_datasets as tfds
import tensorflow as tf

import time
import numpy as np
import matplotlib.pyplot as plt

class Encoder(tf.keras.layers.Layer):
  def __init__(self, num_layers, d_model, num_heads, dff, input_vocab_size,
               maximum_position_encoding, rate=0.1):
    super(Encoder, self).__init__()

    self.d_model = d_model
    self.num_layers = num_layers

    self.embedding = tf.keras.layers.Embedding(input_vocab_size, d_model)
    self.pos_encoding = positional_encoding(maximum_position_encoding,
                                            self.d_model)


    self.enc_layers = [EncoderLayer(d_model, num_heads, dff, rate)
                       for _ in range(num_layers)]

    self.dropout = tf.keras.layers.Dropout(rate)

  def call(self, x, training, mask):

```



```python
  def call(self, x, training, mask):

    seq_len = tf.shape(x)[1]

    # 埋め込みと位置エンコーディングを合算する
    x = self.embedding(x)  # (batch_size, input_seq_len, d_model)
    x *= tf.math.sqrt(tf.cast(self.d_model, tf.float32))
    x += self.pos_encoding[:, :seq_len, :]

    x = self.dropout(x, training=training)

    for i in range(self.num_layers):
      x = self.enc_layers[i](x, training, mask)

    return x  # (batch_size, input_seq_len, d_model)
```



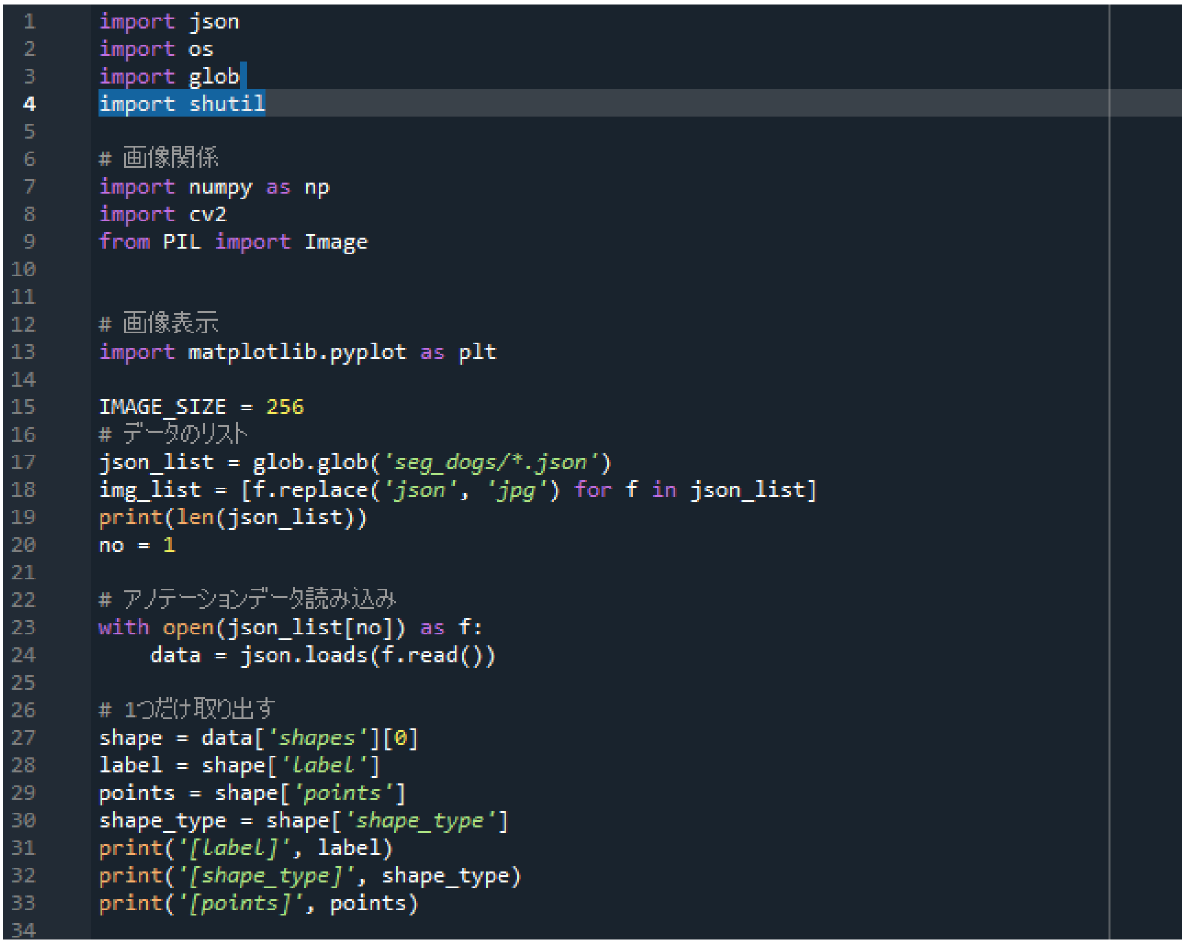
```python
class Decoder(tf.keras.layers.Layer):
  def __init__(self, num_layers, d_model, num_heads, dff, target_vocab_size,
               maximum_position_encoding, rate=0.1):
    super(Decoder, self).__init__()

    self.d_model = d_model
    self.num_layers = num_layers

    self.embedding = tf.keras.layers.Embedding(target_vocab_size, d_model)
    self.pos_encoding = positional_encoding(maximum_position_encoding, d_model)

    self.dec_layers = [DecoderLayer(d_model, num_heads, dff, rate)
                       for _ in range(num_layers)]
    self.dropout = tf.keras.layers.Dropout(rate)

  def call(self, x, enc_output, training,
           look_ahead_mask, padding_mask):

    seq_len = tf.shape(x)[1]
    attention_weights = {}

    x = self.embedding(x)  # (batch_size, target_seq_len, d_model)
    x *= tf.math.sqrt(tf.cast(self.d_model, tf.float32))
    x += self.pos_encoding[:, :seq_len, :]

    x = self.dropout(x, training=training)

    for i in range(self.num_layers):
      x, block1, block2 = self.dec_layers[i](x, enc_output, training,
                                             look_ahead_mask, padding_mask)

      attention_weights['decoder_layer{}_block1'.format(i+1)] = block1
      attention_weights['decoder_layer{}_block2'.format(i+1)] = block2

    # x.shape == (batch_size, target_seq_len, d_model)
    return x, attention_weights
```



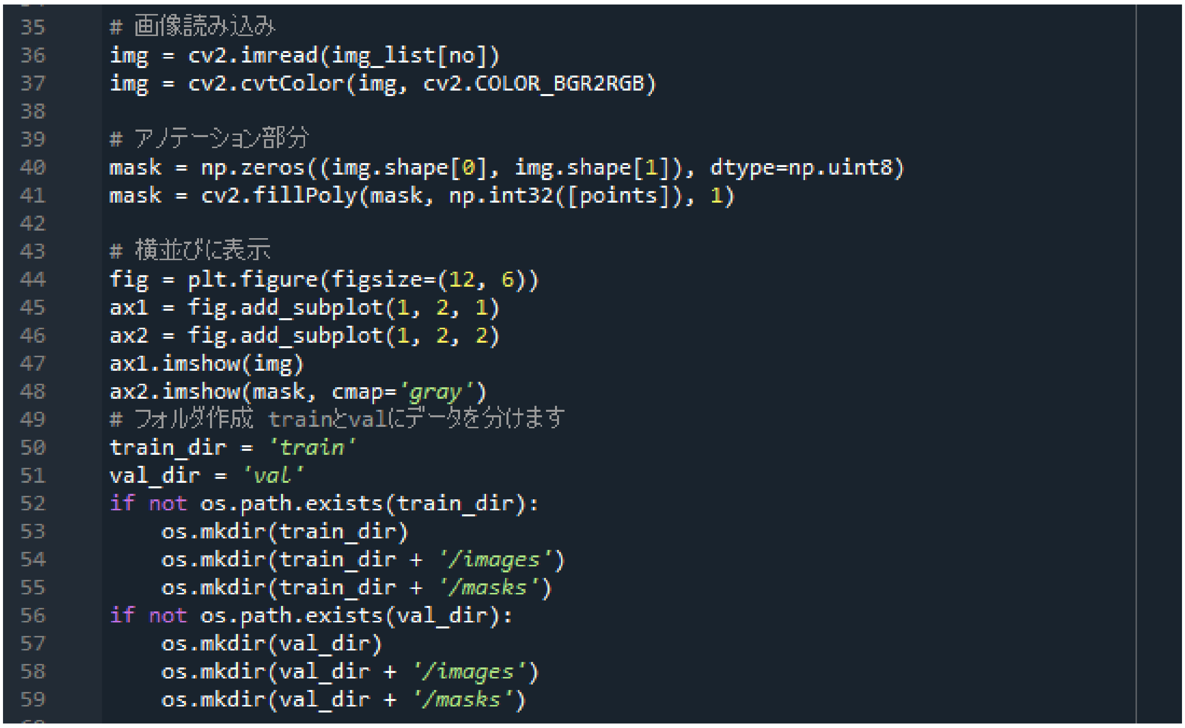
```python
class Transformer(tf.keras.Model):
  def __init__(self, num_layers, d_model, num_heads, dff, input_vocab_size,
               target_vocab_size, pe_input, pe_target, rate=0.1):
    super(Transformer, self).__init__()

    self.encoder = Encoder(num_layers, d_model, num_heads, dff,
                           input_vocab_size, pe_input, rate)

    self.decoder = Decoder(num_layers, d_model, num_heads, dff,
                           target_vocab_size, pe_target, rate)

    self.final_layer = tf.keras.layers.Dense(target_vocab_size)

  def call(self, inp, tar, training, enc_padding_mask,
           look_ahead_mask, dec_padding_mask):

    enc_output = self.encoder(inp, training, enc_padding_mask)  # (batch_size, inp_seq_len, d_model)

    # dec_output.shape == (batch_size, tar_seq_len, d_model)
    dec_output, attention_weights = self.decoder(
        tar, enc_output, training, look_ahead_mask, dec_padding_mask)

    final_output = self.final_layer(dec_output)  # (batch_size, tar_seq_len, target_vocab_size)

    return final_output, attention_weights
```

６．　section6_物体検知_セグメンテーション

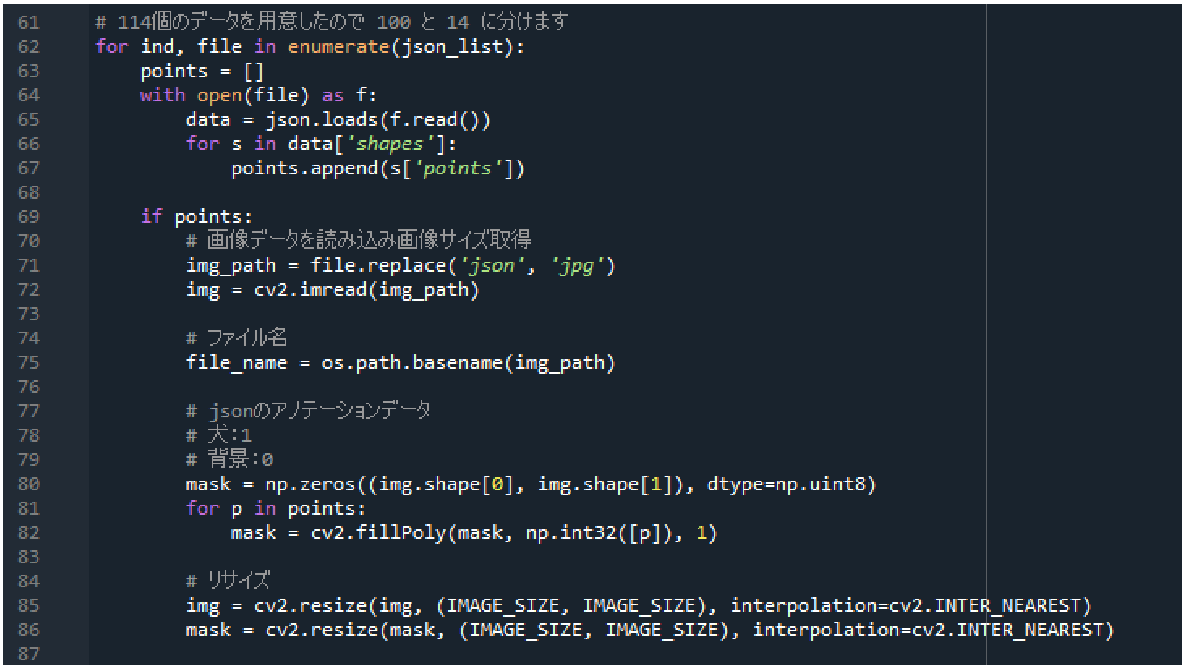
セマンテックセグメンテーションの実装



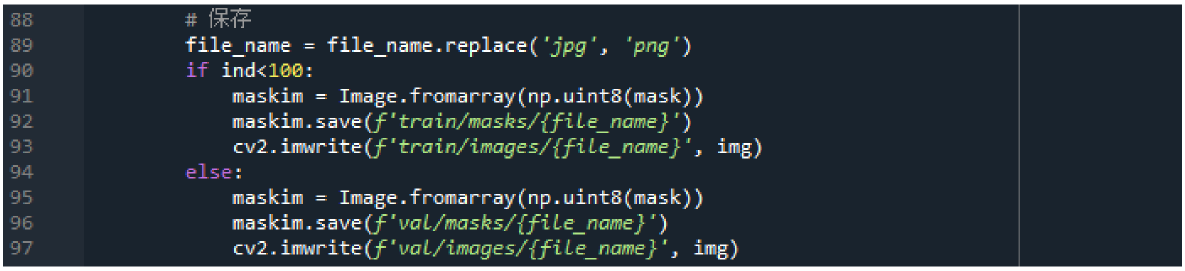
```python
import json
import os
import glob
import shutil

# 画像関係
import numpy as np
import cv2
from PIL import Image


# 画像表示
import matplotlib.pyplot as plt

IMAGE_SIZE = 256
# データのリスト
json_list = glob.glob('seg_dogs/*.json')
img_list = [f.replace('json', 'jpg') for f in json_list]
print(len(json_list))
no = 1

# アノテーションデータ読み込み
with open(json_list[no]) as f:
    data = json.loads(f.read())

# 1つだけ取り出す
shape = data['shapes'][0]
label = shape['label']
points = shape['points']
shape_type = shape['shape_type']
print('[label]', label)
print('[shape_type]', shape_type)
print('[points]', points)

```



```python
# 画像読み込み
img = cv2.imread(img_list[no])
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# アノテーション部分
mask = np.zeros((img.shape[0], img.shape[1]), dtype=np.uint8)
mask = cv2.fillPoly(mask, np.int32([points]), 1)

# 横並びに表示
fig = plt.figure(figsize=(12, 6))
ax1 = fig.add_subplot(1, 2, 1)
ax2 = fig.add_subplot(1, 2, 2)
ax1.imshow(img)
ax2.imshow(mask, cmap='gray')
# フォルダ作成 trainとvalにデータを分けます
train_dir = 'train'
val_dir = 'val'
if not os.path.exists(train_dir):
    os.mkdir(train_dir)
    os.mkdir(train_dir + '/images')
    os.mkdir(train_dir + '/masks')
if not os.path.exists(val_dir):
    os.mkdir(val_dir)
    os.mkdir(val_dir + '/images')
    os.mkdir(val_dir + '/masks')
```



```python
# 114個のデータを用意したので 100 と 14 に分けます
for ind, file in enumerate(json_list):
    points = []
    with open(file) as f:
        data = json.loads(f.read())
        for s in data['shapes']:
            points.append(s['points'])

    if points:
        # 画像データを読み込み画像サイズ取得
        img_path = file.replace('json', 'jpg')
        img = cv2.imread(img_path)

        # ファイル名
        file_name = os.path.basename(img_path)

        # jsonのアノテーションデータ
        # 犬:1
        # 背景:0
        mask = np.zeros((img.shape[0], img.shape[1]), dtype=np.uint8)
        for p in points:
            mask = cv2.fillPoly(mask, np.int32([p]), 1)

        # リサイズ
        img = cv2.resize(img, (IMAGE_SIZE, IMAGE_SIZE), interpolation=cv2.INTER_NEAREST)
        mask = cv2.resize(mask, (IMAGE_SIZE, IMAGE_SIZE), interpolation=cv2.INTER_NEAREST)

```



```python
        # 保存
        file_name = file_name.replace('jpg', 'png')
        if ind<100:
            maskim = Image.fromarray(np.uint8(mask))
            maskim.save(f'train/masks/{file_name}')
            cv2.imwrite(f'train/images/{file_name}', img)
        else:
            maskim = Image.fromarray(np.uint8(mask))
            maskim.save(f'val/masks/{file_name}')
            cv2.imwrite(f'val/images/{file_name}', img)
```