Task deadline: 11:59pm 18th May 2024

Bottomline: Just pass the grader

# Task details

## First Task

Tune the existing classifier to a accuracy greater than 90%.

Use the “data” folder for this part of the project. Add to the existing code to achieve the above goal. This might require several tricks:

* Input normalization
* Residual blocks
* Dropout
* Data augmentations (Both geometric and color augmentations are important. Be aggressive here. Different levels of supertux have radically different lighting.)
* Weight regularization
* Early stopping

**Work to fill in some codes in *utils.py, models.py, train\_cnn.py***

**Test your model with**

*python3 -m grader homework*

## Second Task

Dense prediction (semantic segmentation)

**In the second part, we want to make your CNN fully convolutional. Instead of predicting a single output per image, we want now predict an output per pixel.**

We will use a dense prediction dataset (dense\_data), where all images have a resolution, and the labels are of the same size. We have 5 labels here: background, kart, track, bomb/projectile, pickup/nitro. We merged the last two classes, as they already have very few labeled instances. The class distribution for those labels is quite bad, background and track make up 96% of the labeled area!

**To see some examples from the dense dataset run:**

*python3 -m homework.utils*

**FCN design**

Design your FCN by writing the model in *models.py*. Make sure to use only convolutional operators, pad all of them correctly and match strided operators with up-convolutions. Use skip and residual connections.

Make sure your FCN handles an arbitrary input resolution and produces an output of the same shape as the input. Use output\_padding=1 if needed. Crop the output if it is too large.

**Test your model with:**

*python3 -m grader homework*

**Work to fill in some codes in *utils.py, models.py***

**FCN Training**

To train your FCN you'll need to modify your CNN training code a bit. First, you need to use the DenseSuperTuxDataset. This dataset accepts data augmentation parameters transform. Most standard data augmentation in torchvision do not directly apply to dense labeling tasks. We thus provide you with a smaller subset of useful augmentations that properly work with a pair of image and label in *dense\_transforms.py*.

You will need to use the same bag of tricks as for classification to make the FCN train well.

Since the training set has a large class imbalance, it is easy to cheat in a pixel-wise accuracy metric. Predicting only track and background gives a accuracy. We additionally measure the Intersection-over-Union evaluation metric. This is a standard semantic segmentation metric that penalizes largely imbalanced predictions. This metric is harder to cheat, as it computes \frac{\text{true positives}}{\text{true positives} + \text{false positives} + \text{false negatives}}. You might need to change the class weights of your torch.nn.CrossEntropyLoss, although our master solution did not require this. You can compute the IoU and accuracy using the ConfusionMatrix class.

**Test your model with:**

*python3 -m grader homework*

Work to fill in some codes in utils.py, models.py, train\_fcn.py