Βαθιά Μάθηση και Ανάλυση Πολυμεσικών Δεδομένων

Ευαγγελία Κυριακοπούλου

ΑΕΜ : 110

Contents:

1. Introduction

Από την βιβλιοθήκη gymnasium για την εργασία αυτή επιλέχθηκε το παιχνίδι Mario Bros της Atari. Την εργασία αυτή την προσεγγίσαμε με χρήση Βαθιάς Ενισχυτής Μάθησης (Deep Reinforcement Learning). Μέσω της βιβλιοθήκης Stable Baseline 3 και του PPO αλγορίθμου εκπαιδεύτηκε ο agent και συγκεκριμένα για το CNNPolicy. Το μοντέλο αρχικά εκπαιδεύτηκε με τις default τιμές του και στην συνέχεια προστέθηκε σε αυτό ένα Custom Συνελικτικό Δίκτυο και ακολούθησαν κάποιοι πειραματισμοί με προσθέσεις και αλλαγές πάνω στο δίκτυο αυτό για την βελτίωση του.

2. Importing necessary libraries

Το πρόγραμμα υλοποιήθηκε στη γλώσσα προγραμματισμού Python με χρήση των παρακάτω βιβλιοθηκών :

```
!apt-get update && apt-get install ffmpeg freeglut3-dev xvfb  # For visualization
!pip install "stable-baselines3[extra]>=2.0.0a4"

import stable_baselines3

print(f"{stable_baselines3.__version__=}")

import os
import gymnasium as gym
import tensorflow as tf
import imageio
import numpy as np
import torch as th
import torch.nn as nn
from stable_baselines3 import PPO
from stable_baselines3.common.evaluation import evaluate_policy
from stable_baselines3.common.env_util import make_atari_env
from stable_baselines3.common.vec_env import VecFrameStack
from stable_baselines3.common.callbacks import CallbackList, CheckpointCallback, EvalCallback,StopTrainingOnNoModelImprovement
from stable_baselines3.common.torch_layers import BaseFeaturesExtractor

from stable_baselines3.common.vec_env import DummyVecEnv, SubprocVecEnv
from stable_baselines3.common.env_util import make_vec_env
from stable_baselines3.common.utils import set_random_seed
%load_ext tensorboard
%tensorboard --logdir logs
```

Figure 1 Φόρτωση Βιβλιοθηκών

3. PPO Default Model

Για την PPO Default εκπαίδευση του μοντέλου : το learning rate παρέμεινε σταθερό με τιμή ίση με 0.01, τα steps στα οποία εκπαιδεύτηκε ήταν 100K, χρησιμοποιήθηκαν δύο environments και χωρίστηκαν σε training & evaluation έτσι ώστε να έχουμε καλύτερη εικόνα για την αξιολόγηση του μοντέλου. Το evaluation γινόταν κάθε 1000 βήματα. Παρακάτω βλέπουμε κάποια από τα αποτελέσματα του:

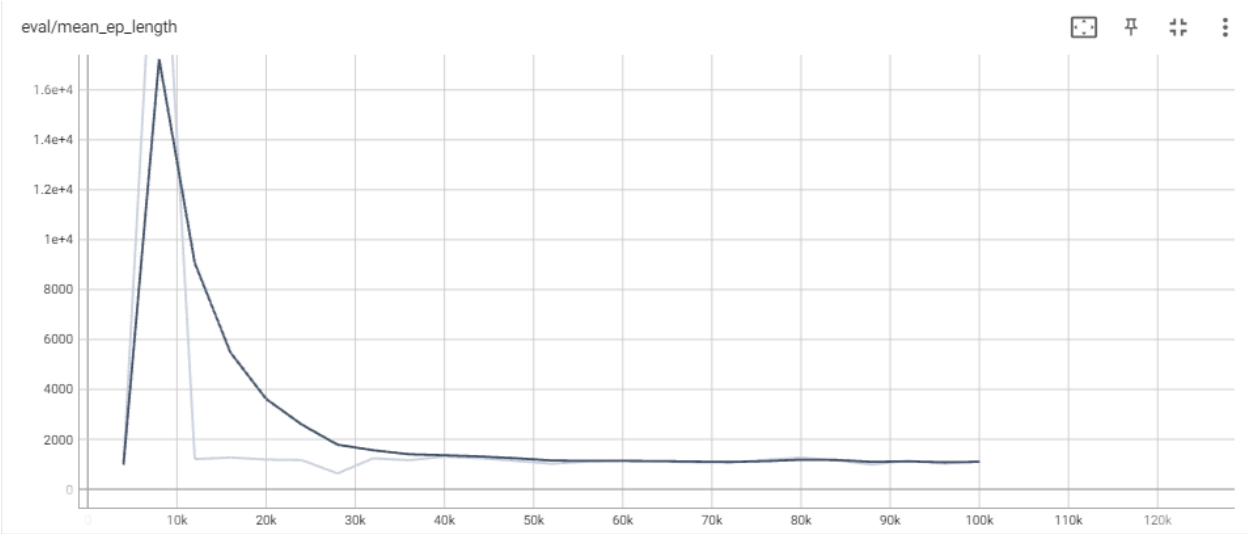

Figure 2 eval/mean_ep_length

Από την *Figure 2* εικόνα παρατηρούμε ότι ο Mario παραμένει περισσότερη ώρα ζωντανός στην πίτσα χωρίς να χάσει μέχρι και λίγο πριν τα πρώτα 10K βήματα. Στην συνέχεια με βάση το διάγραμμα καταλαβαίνουμε ότι δεν μπορεί να ανταπεξέλθει και παραμένει ελάχιστα ζωντανός.

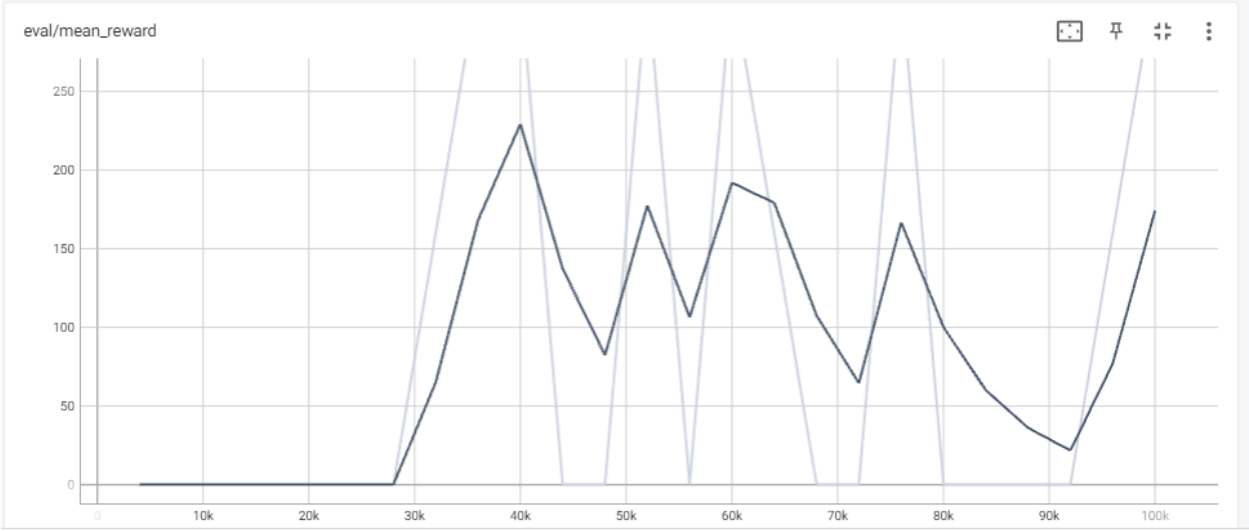

Figure 3 eval/mean_reward

Με βάση την *Figure 3* εικόνα παρατηρούμε ότι το reward του μέχρι και τα 30K βήματα παραμένει μηδενικό, ενώ όσο προχωράει η εκπαίδευση μεταξύ 30-40K βήματα αποκτά την υψηλότερη επιβράβευση (220)και στη συνέχεια μέχρι και το τέλος των 100K βημάτων, ενώ λαμβάνει επιβράβευση δεν ξεπερνά την υψηλότερη τιμή που αποκτήθηκε στα αρχικά βήματα. Με βάση τα δύο διαγράμματα αυτά μπορούμε να πούμε ότι μετά τα 10K βήματα η διάρκεια ζωής του στο παιχνίδι είναι λιγότερη παρόλα αυτά με το πέρας των 30K και όσο καταφέρνει να

μείνει ζωντανός δίχως να χάσει πέφτοντας πάνω σε κάποιο εμπόδιο(χελωνάκι, σφαίρα) κερδίζει rewards.

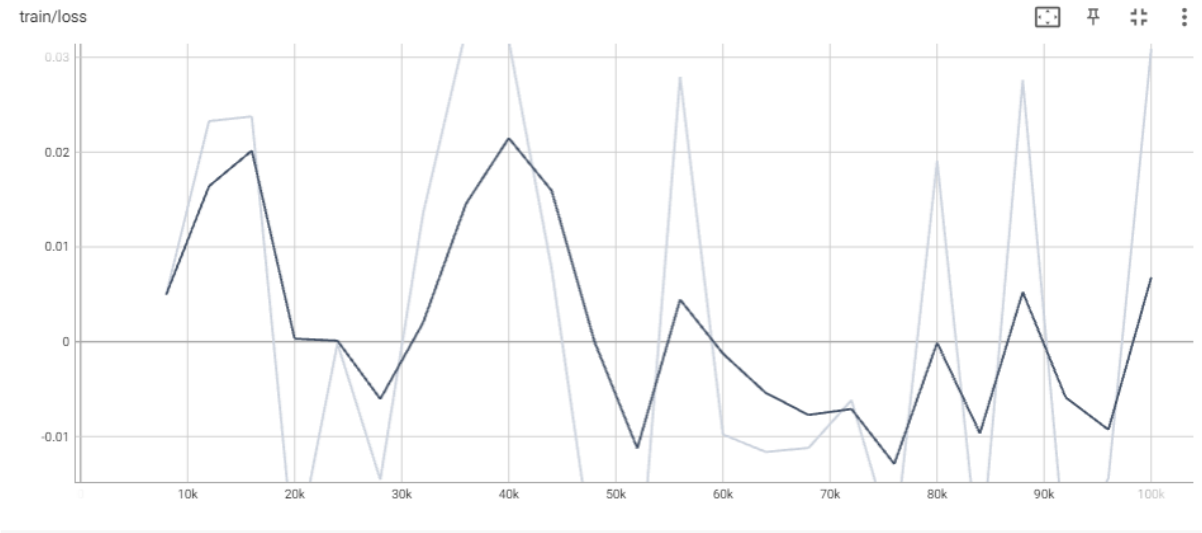

Figure 4 train/loss

Από την *Figure 4* βλέπουμε τα losses να έχουν διάφορες διακυμάνσεις θετικές και αρνητικές και να μην είναι σταθερά, αυτό ίσως αυτό οφείλεται στον αριθμό βημάτων που έχουμε επιλέξει για την εκπαίδευση. Τα 100K είναι ένας μικρός αριθμός για μια τέτοια εκπαίδευση για να παρατηρήσουμε πτώση ή σταθεροποίηση στις απώλειες.

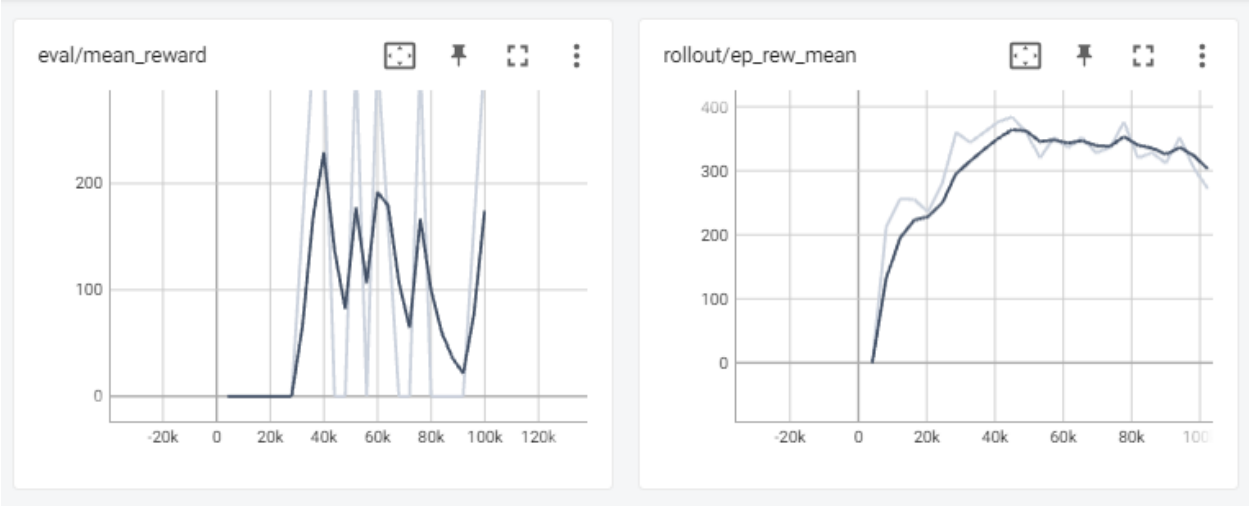

Figure 5 eval/train rewards

Στη συνέχεια το ίδιο μοντέλο με τις ίδιες default τιμές εκπαιδεύτηκε στα 200K βήματα κάνοντας χρήση όμως αυτή τη φορά 6 περιβάλλοντα αντί για 2. Τα αποτελέσματα του ήταν σαφώς καλύτερα και φαίνονται παρακάτω.

Figure 6 eval/mean_ep_length - eval/mean_reward

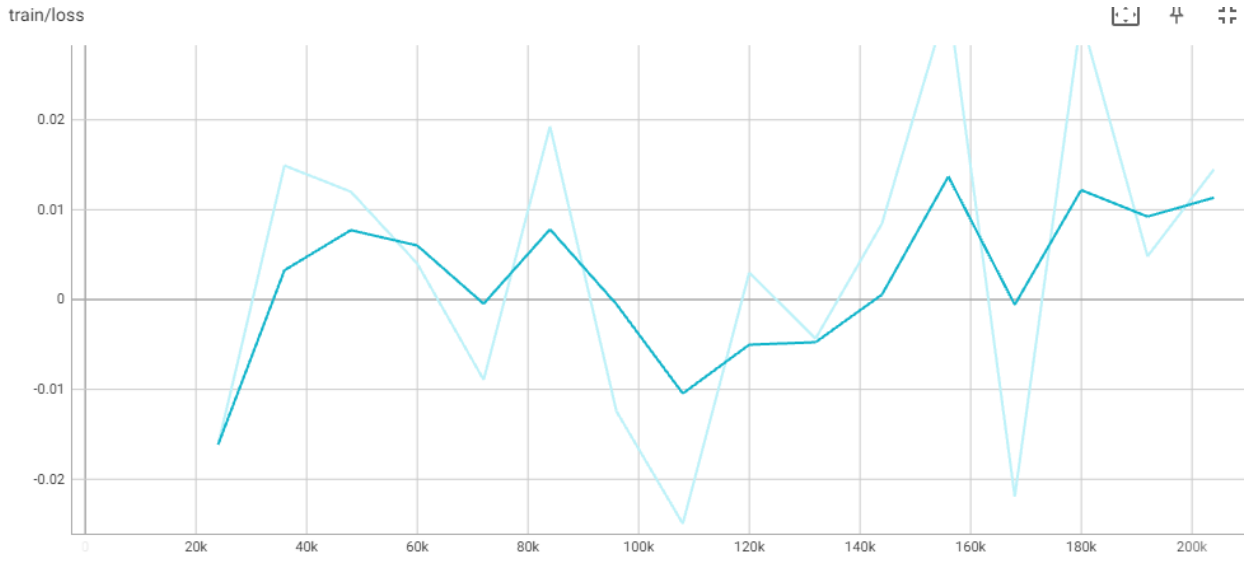

Figure 7 Train/loss

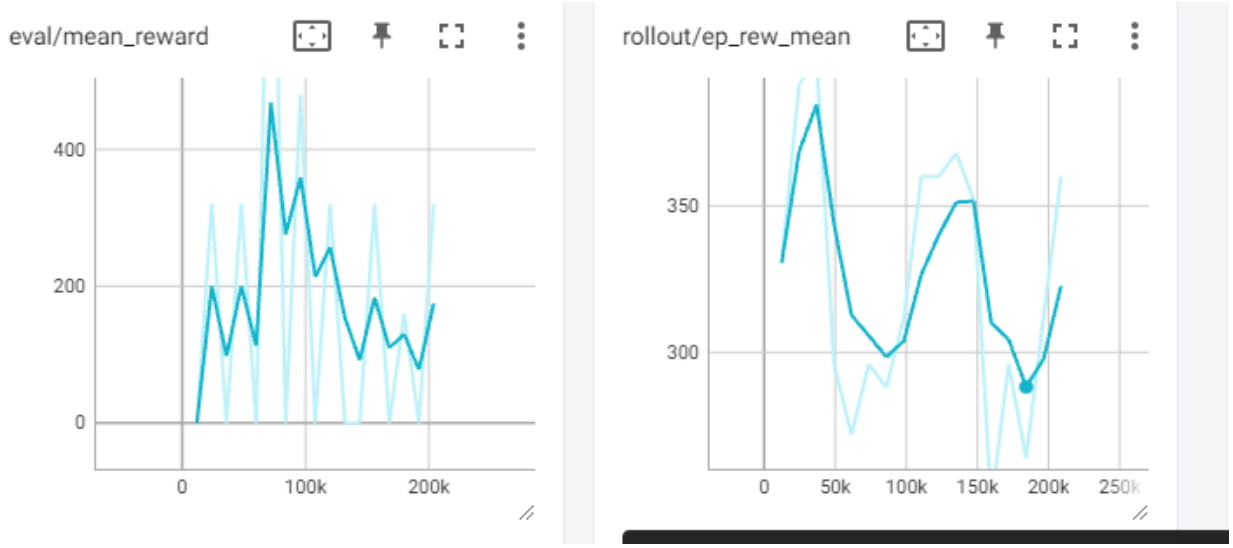

Figure 8 eval/train rewards

Από την *Figure 6* εικόνα παρατηρούμε ότι ο Mario παραμένει περισσότερη ώρα ζωντανός στην πίτσα χωρίς να χάσει μέχρι και λίγο πριν τα πρώτα 150K βήματα σε αντίθεση με την προηγούμενη εκπαίδευση που η διάρκεια ζωής του κρατούσε μέχρι τα 10K πρώτα βήματα και μετά έχανε και υπήρχε μόνο πτώση στο διάγραμμα ζωής του. Παρατηρείτε επίσης μεγάλη διαφορά στην επιβράβευση (reward) την οποία λαμβάνει καθώς από το παραπάνω εικόνα η

μέγιστη ανταμοιβή που παίρνει είναι 420 δηλαδή η διπλάσια από την προηγούμενη εκπαίδευση του. Το διάγραμμα της απώλειας *Figure 7* θα μπορούσαμε να πούμε ότι είναι αρκετά βελτιωμένο καθώς οι μεταβάσεις γίνονται πιο φυσικά και οι τιμές κυμαίνονται και αυτές σ ένα καλύτερο εύρος από [-0.015, ~0.015] σε αντίθεση με την εκπαίδευση των 100K βημάτων που οι τιμές βρίσκονται στο εύρος [-0.012,0.022] και οι μεταβάσεις δεν γίνονται τόσο smooth. Επομένως η σύγκριση που θα δούμε στο τέλος θα γίνει με βάση το default μοντέλο που εκπαιδεύτηκε στα 200K βήματα και 6 περιβάλλοντα.

4 . Try For Fine Tuning

Στην συνέχεια έγινε δοκιμή προσθέτοντας ένα Συνελικτικό Δίκτυο το οποίο έχει την ακόλουθη αρχιτεκτονική. Το CNN περιέχει τα ακόλουθα επίπεδα:

Conv2d: έχουμε ένα 2D συνελικτικό επίπεδο με έξοδο 32 καναλιών, χρησιμοποιώντας μέγεθος πυρήνα 3x3 . Ακολουθεί η συνάρτηση ενεργοποίησης ReLU που εφαρμόζεται μετά την πρώτη συνελικτική στρώση. Έπειτα έχουμε ένα δεύτερο 2D συνελικτικό επίπεδο με είσοδο 32 καναλιών και έξοδο 64 καναλιών χρησιμοποιώντας πυρήνα 3x3 και καλείται πάλι η ίδια συνάρτηση ενεργοποίησης ReLU, η οποία εφαρμόζεται μετά το δεύτερο συνελικτικό στρώμα. Ακολουθεί το στρώμα Flatten το οποίο θα κάνει μετατροπή της εξόδου σε 1D. Παρακάτω βλέπουμε και το σχετικό κώδικα :

```python
class CustomCNN(BaseFeaturesExtractor):

    def __init__(self, observation_space: gym.spaces.Box, features_dim: int=128):
        super(CustomCNN, self).__init__(observation_space, features_dim)

        n_input_channels = observation_space.shape[0]
        self.cnn = nn.Sequential(
            nn.Conv2d(n_input_channels,32, kernel_size=3, stride=1, padding=0),
            nn.ReLU(),
            nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=0),
            nn.ReLU(),
            nn.Flatten(),
        )

        with th.no_grad():
            n_flatten = self.cnn(
                th.as_tensor(observation_space.sample()[None]).float()
            ).shape[1]

        self.linear = nn.Sequential(nn.Linear(n_flatten, features_dim), nn.ReLU())

    def forward(self, observations: th.Tensor) -> th.Tensor:
        return self.linear(self.cnn(observations))

policy_kwargs = dict(
    features_extractor_class=CustomCNN,
    features_extractor_kwargs=dict(features_dim=128)
)
```

Figure 9 Custom CNN

Η εκπαίδευση πραγματοποιήθηκε με τις ίδιες παραμέτρους learning_rate=0.01, environments=2, βήματα εκπαίδευσης 100K και evaluation κάθε 1000 βήματα.

Παρακάτω έχουμε τα ακόλουθα αποτελέσματα :

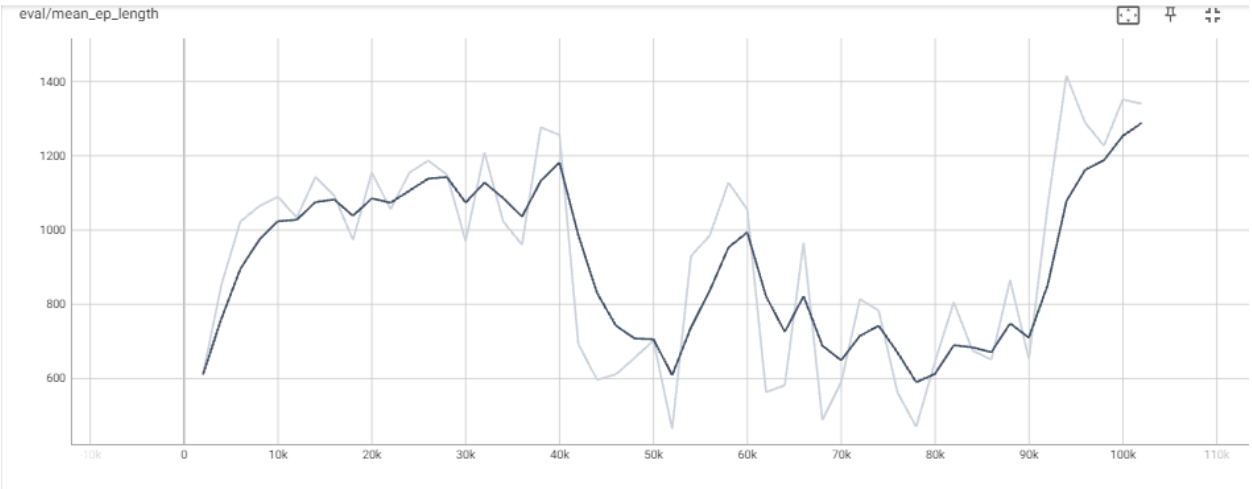

Figure 10 eval/mean_ep_length

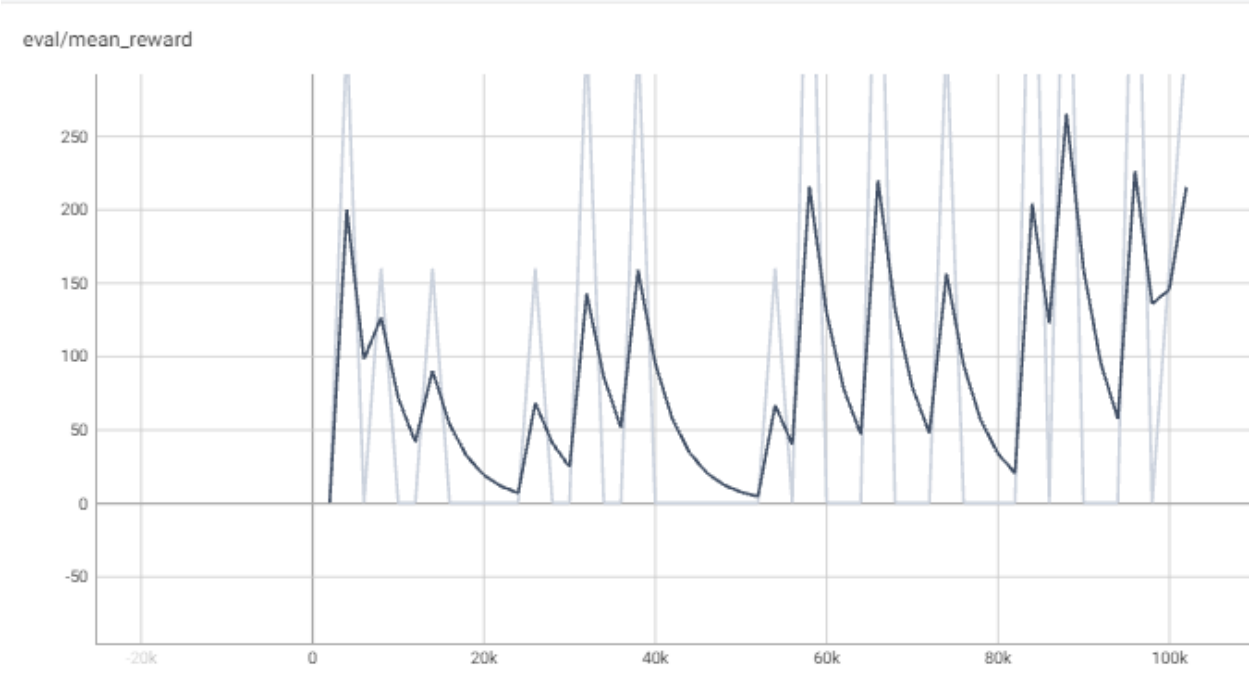

Figure 11 eval/mean_reward

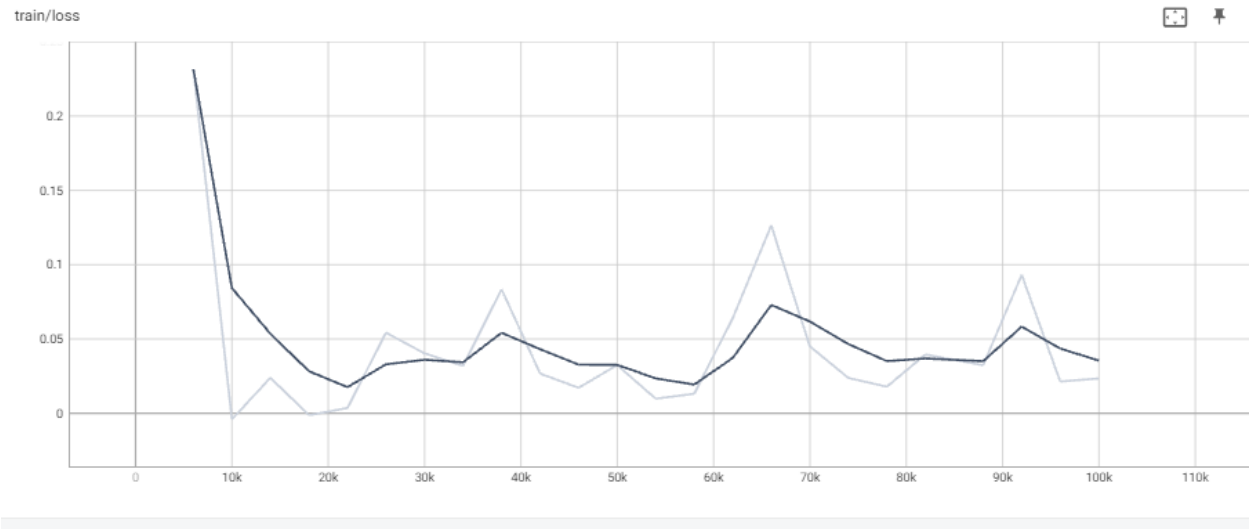

Figure 12 Train/loss

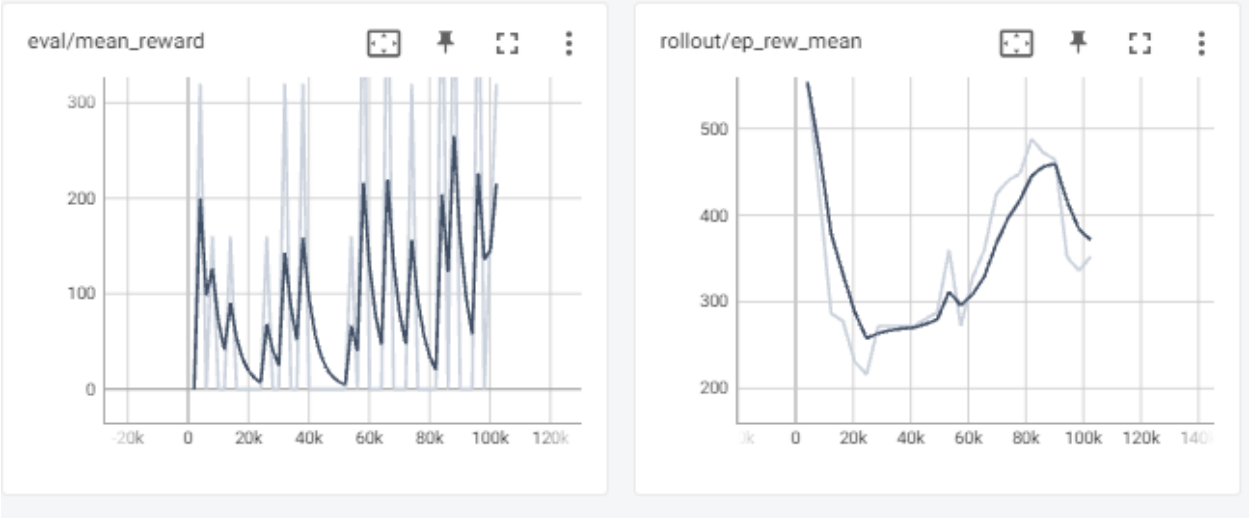

Figure 13 eval/train rewards

Συγκριτικά λοιπόν με το μοντέλο που εκπαιδεύτηκε πρώτα με τα τις default τιμές του PPO στα 100 βήματα, βλέπουμε ότι το τωρινό μοντέλο αποδίδει καλύτερα. Αυτό το συμπέρασμα βγαίνει από τα διαγράμματα που βλέπουμε παραπάνω, καθώς η διάρκεια ζωής του agent είναι σαφώς μεγαλύτερη κατά την διάρκεια εκτέλεσης του παιχνιδιού, τα reward που λαμβάνει είναι εξίσου μεγαλύτερα, πετυχαίνοντας νέο καλύτερο ρεκόρ με τιμή 260. Επιπλέον με βάση το διάγραμμα των απωλειών παρατηρούμε ότι ξεκινάει με loss λίγο μεγαλύτερη του 0.02 και όσο αυξάνονται τα βήματα της εκπαίδευσης η απώλεια θα λέγαμε πως σταθεροποιείται μεταξύ του μηδενός και του 0.01.

Η επόμενη εκπαίδευση έγινε με μια διαφορετική αρχιτεκτονική στο συνελικτικό δίκτυο η οποία φαίνεται παρακάτω μαζί με τα σχετικά αποτελέσματα της εκπαίδευσης :

```
# Neural network for predicting action values ----arxiko cnn
class CustomCNN(BaseFeaturesExtractor):

    def __init__(self, observation_space: gym.spaces.Box, features_dim: int=128):
        super(CustomCNN, self).__init__(observation_space, features_dim)

        n_input_channels = observation_space.shape[0]
        self.cnn = nn.Sequential(
            nn.Conv2d(n_input_channels,128, kernel_size=3, stride=1, padding=0),
            nn.ReLU(),
            nn.Conv2d(128, 64, kernel_size=3, stride=1, padding=0),
            nn.ReLU(),
            nn.Flatten(),
        )

        # Compute shape by doing one forward pass
        with th.no_grad():
            n_flatten = self.cnn(
                th.as_tensor(observation_space.sample()[None]).float()
            ).shape[1]

        self.linear = nn.Sequential(nn.Linear(n_flatten, features_dim), nn.ReLU())

    def forward(self, observations: th.Tensor) -> th.Tensor:
        return self.linear(self.cnn(observations))

policy_kwargs = dict(
    features_extractor_class=CustomCNN,
    features_extractor_kwargs=dict(features_dim=128)
)
```

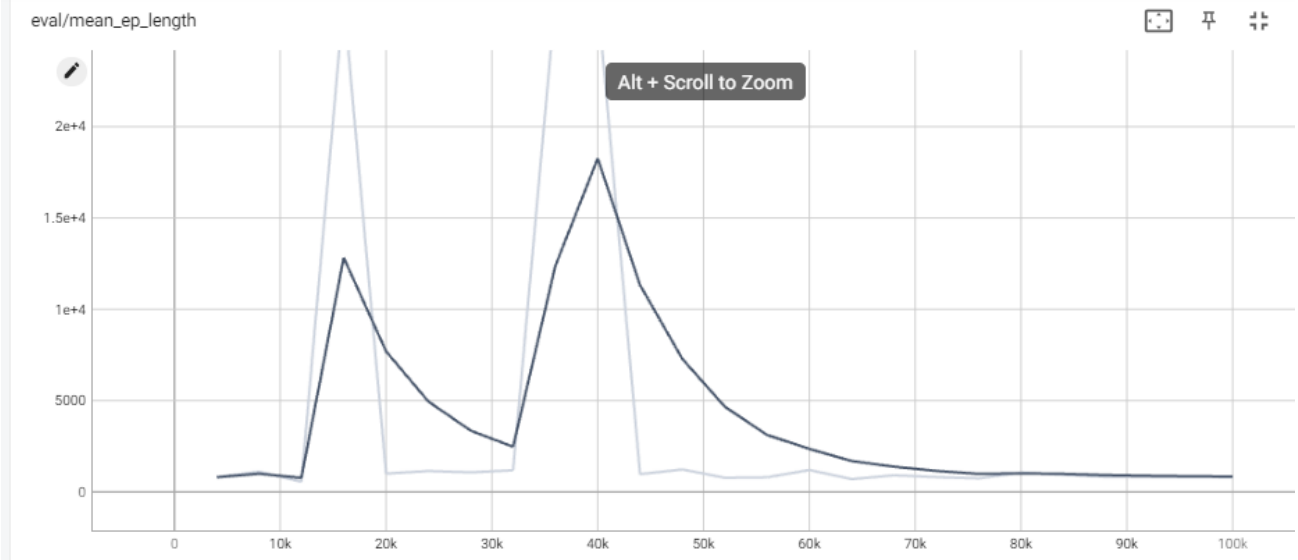

Figure 14 eval/mean_ep_length

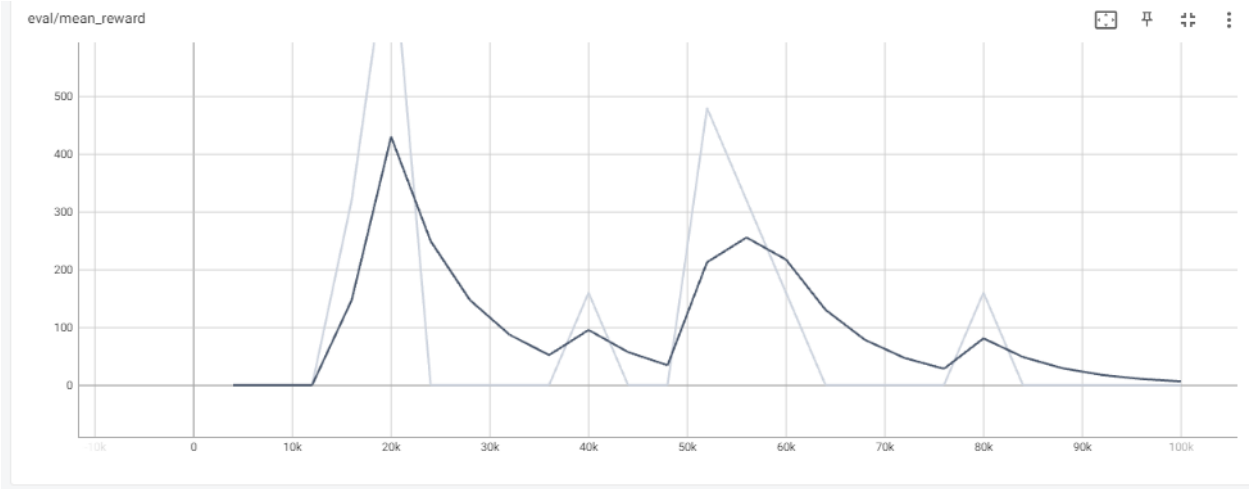

Figure 15 eval/mean_reward

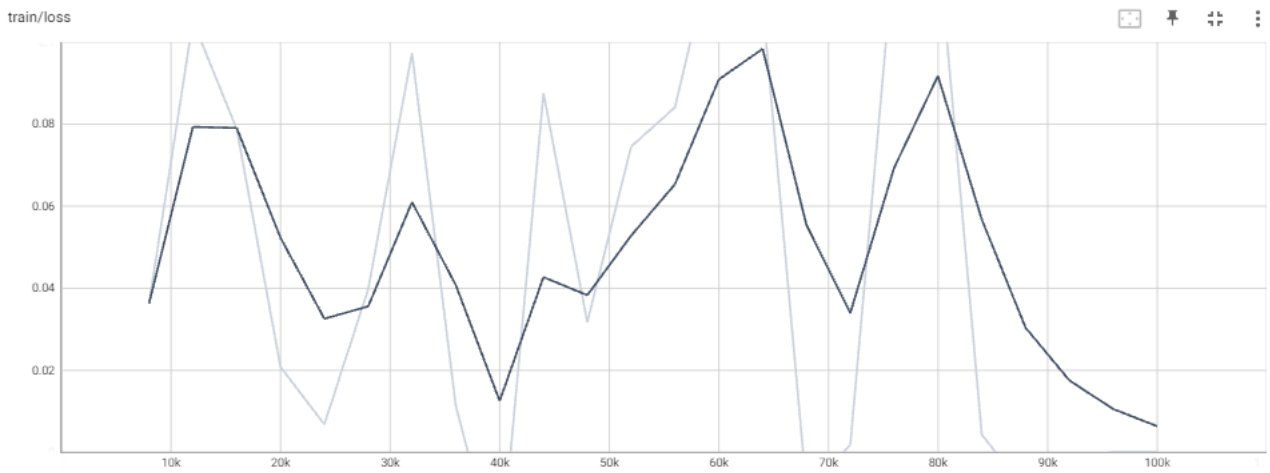

Figure 16 train/loss

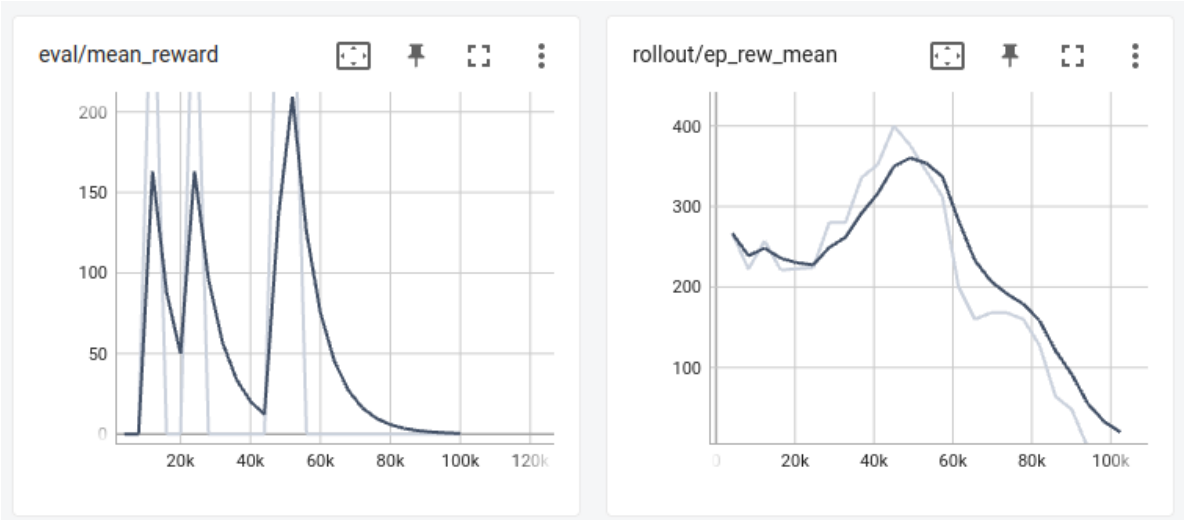

Figure 17 eval/train rewards

Το παραπάνω μοντέλο με την συγκεκριμένη αρχιτεκτονική που δοκιμάστηκε έδωσε καλύτερα αποτελέσματα σε σχέση με το προηγούμενο μοντέλο. Εάν παρατηρήσουμε τα διαγράμματα το reward που λαμβάνει κατά την διάρκεια της ζωής του agent στην πλατφόρμα παιχνιδιού είναι σχεδόν 100 μονάδες πάνω. Στις υπόλοιπες μετρικές δεν απέδωσε τόσο καλά αποτελέσματα με βάση τα διαγράμματα που πάρθηκαν από το tesnorboard. Για το λόγο αυτό προστέθηκαν δυο Dense Layers που δεν υπήρχαν, αυξήθηκαν τα environments εκπαίδευσης από δύο σε έξι όπως επίσης και ο αριθμός των βημάτων από 100K στα 200K. Παρακάτω έχουμε τα τελευταία αποτελέσματα όπως επίσης και την σύγκριση με το PPO μοντέλο με τις default τιμές εκπαίδευσης επίσης στα 200K βήματα.

```python
# Neural network for predicting action values
class CustomCNN(BaseFeaturesExtractor):

    def __init__(self, observation_space: gym.spaces.Box, features_dim: int=128):
        super(CustomCNN, self).__init__(observation_space, features_dim)
        # CxHxW images (channels first)
        n_input_channels = observation_space.shape[0]
        self.cnn = nn.Sequential(
            nn.Conv2d(n_input_channels,128, kernel_size=3, stride=1, padding=0),
            nn.ReLU(),
            nn.Conv2d(128, 64, kernel_size=3, stride=1, padding=0),
            nn.ReLU(),
            nn.Flatten(),
            nn.Linear(409600,128),
            nn.Linear(128,64)
        )

        # Compute shape by doing one forward pass
        with th.no_grad():
            n_flatten = self.cnn(
                th.as_tensor(observation_space.sample()[None]).float()
            ).shape[1]

        self.linear = nn.Sequential(nn.Linear(64, features_dim), nn.ReLU())

    def forward(self, observations: th.Tensor) -> th.Tensor:
        return self.linear(self.cnn(observations))

policy_kwargs = dict(
    features_extractor_class=CustomCNN,
    features_extractor_kwargs=dict(features_dim=128)
)
```

Figure 18 Νέα Αρχιτεκτονική Δικτύου με προσθήκη Dense Layers

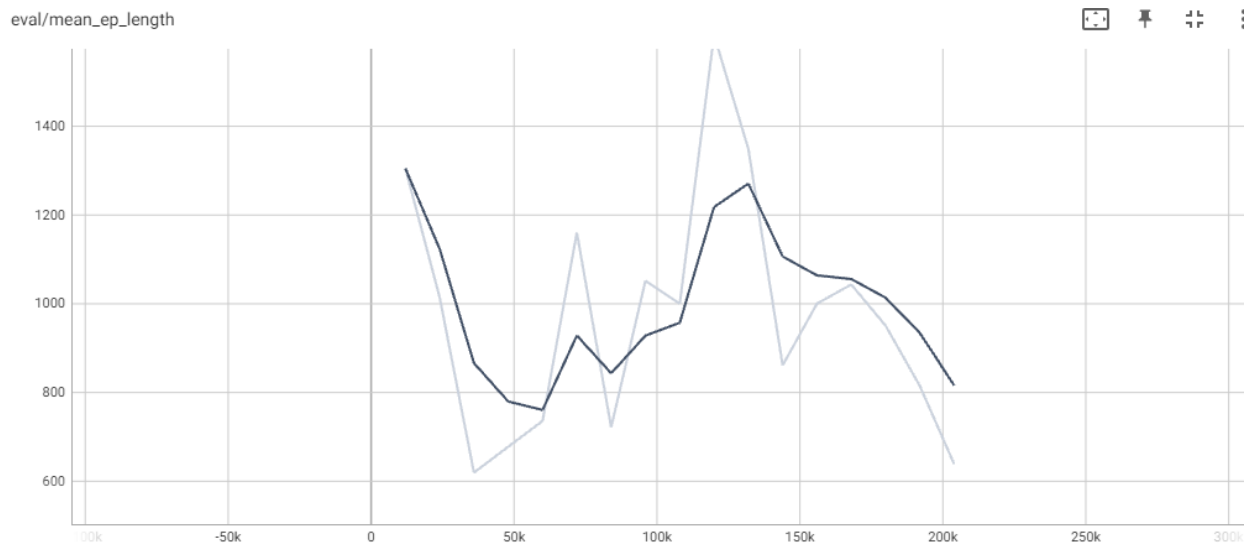

Figure 19 eval/mean_ep_length

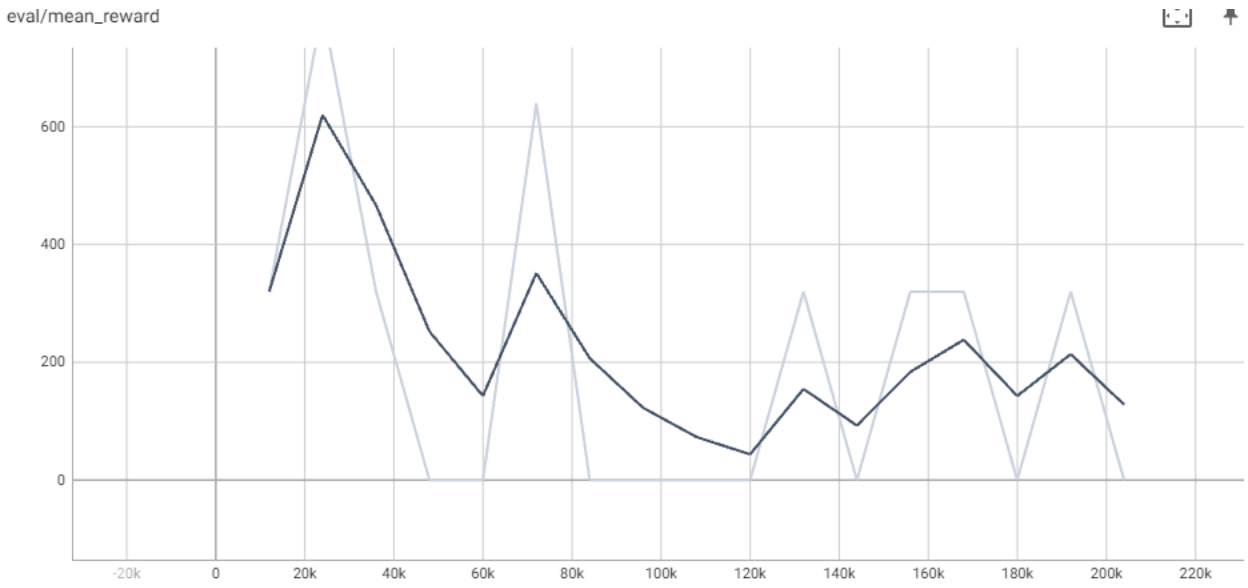

Figure 20 eval/mean_reward

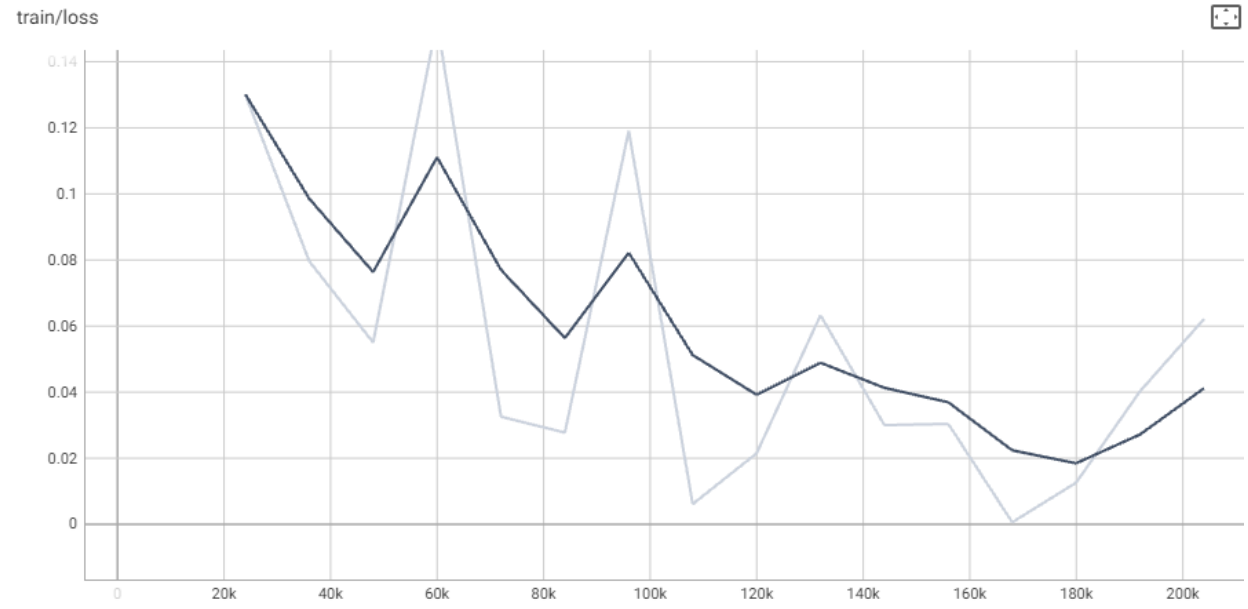

Figure 21 train/loss

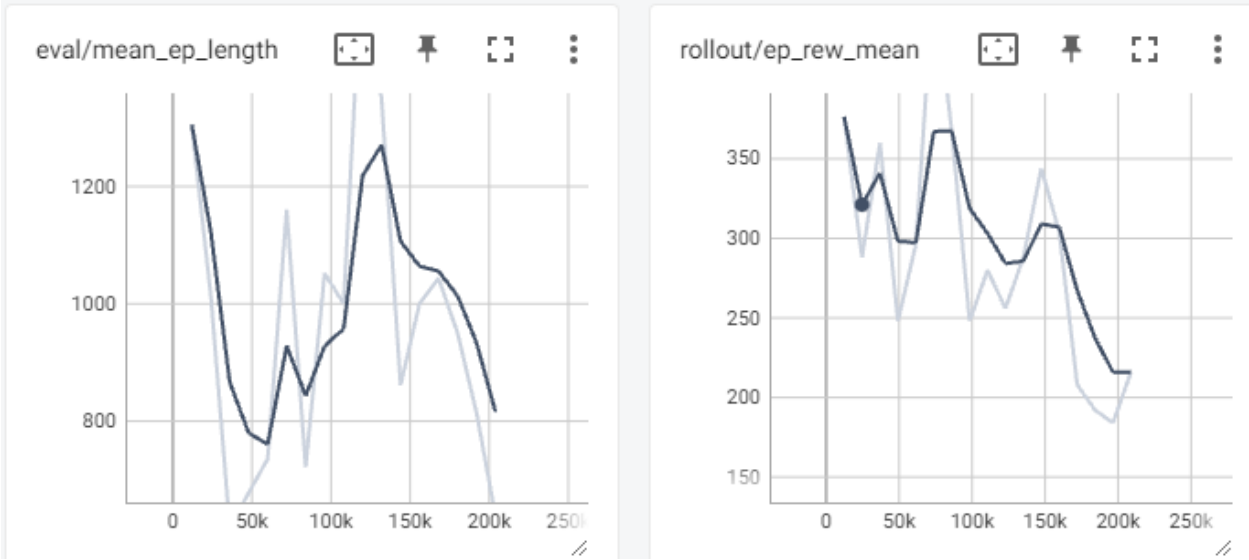

Figure 22 eval/train rewards

Η προσθήκη των Dense Layers δεν προσέδωσε θετική διαφορά στα αποτελέσματα ίσα ίσα οι περισσότερες μετρικές απέδωσαν χειρότερα. Με την χρήση πολύ παραπάνω βημάτων ίσως 1M+ τα αποτελέσματα να είχαν εμφανή διαφορά. Λόγω έλλειψης μνήμης του υπολογιστή και χρόνου οι πειραματισμοί που μπόρεσαν να γίνουν ήταν μέχρι και αυτό το επίπεδο. Παρακάτω ακολουθεί η σύγκριση των μοντέλων PPO-default τιμών με Custom CNN στα 200K βήματα.

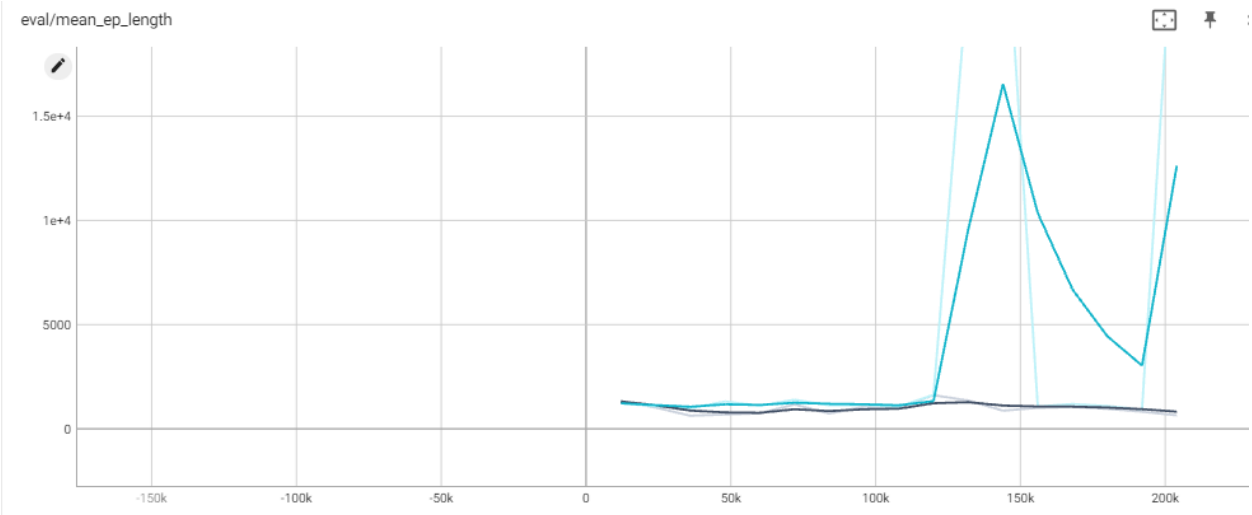

Figure 23 eval/mean_ep_length

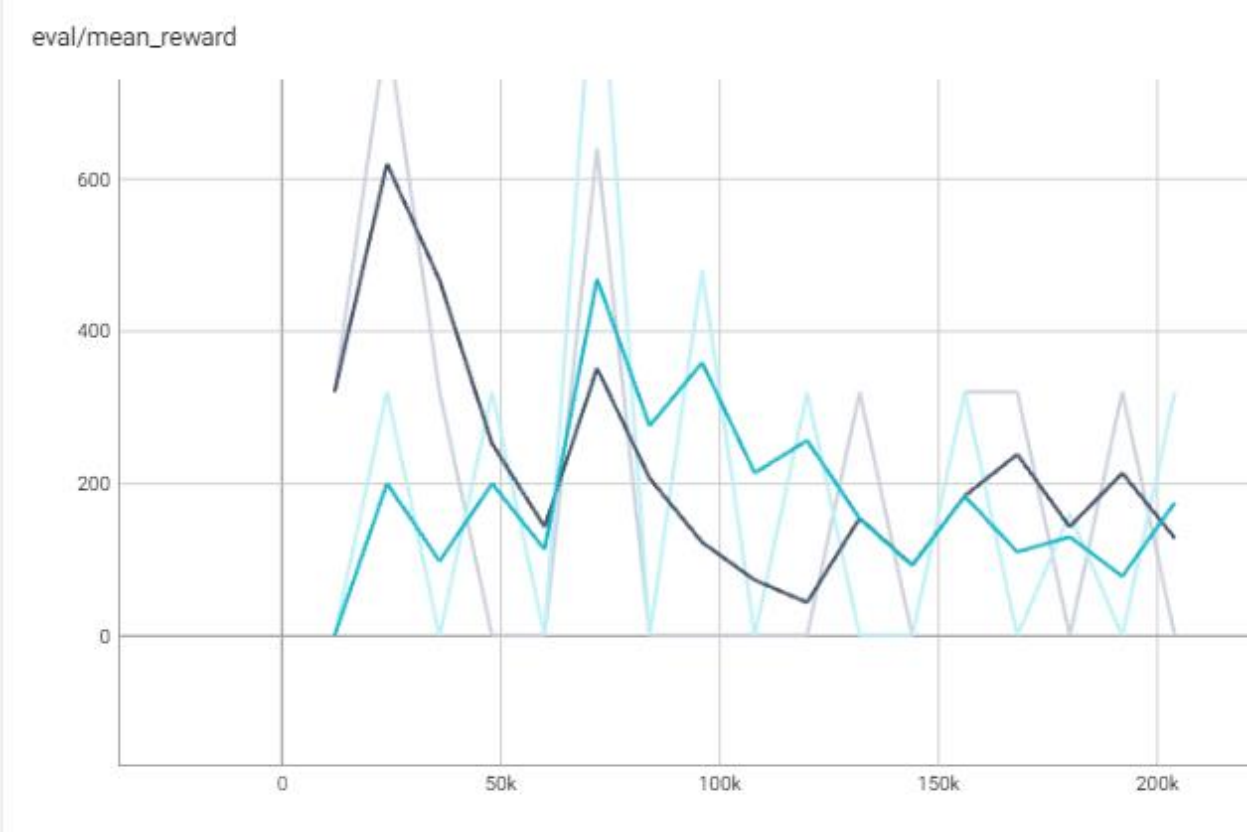

Figure 24 eval/mean_reward

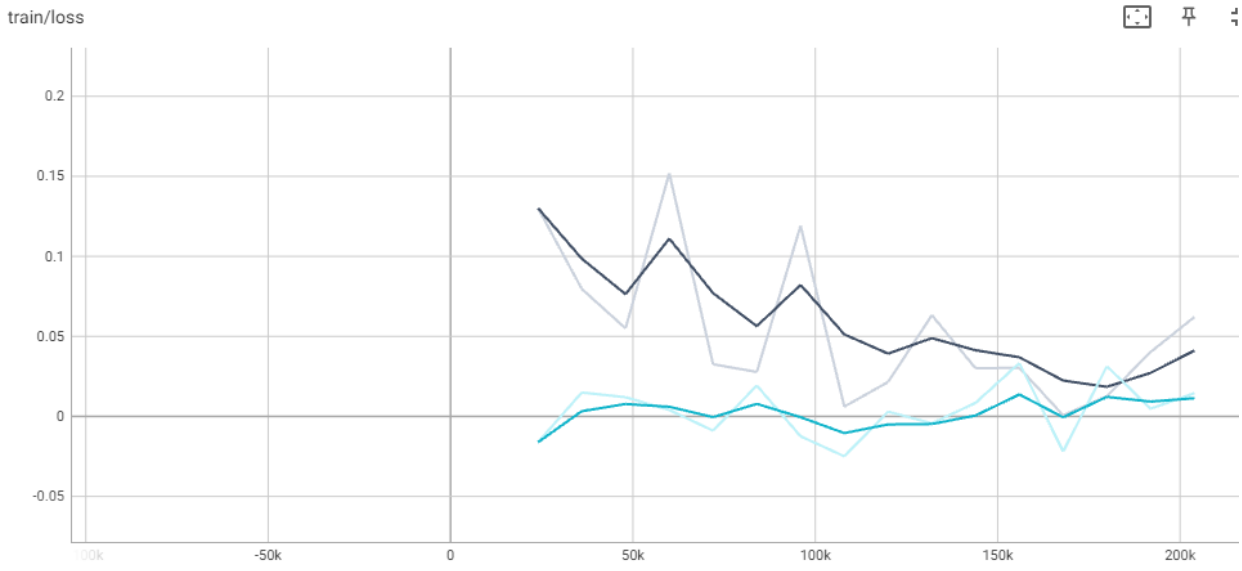

Figure 25 Train/loss

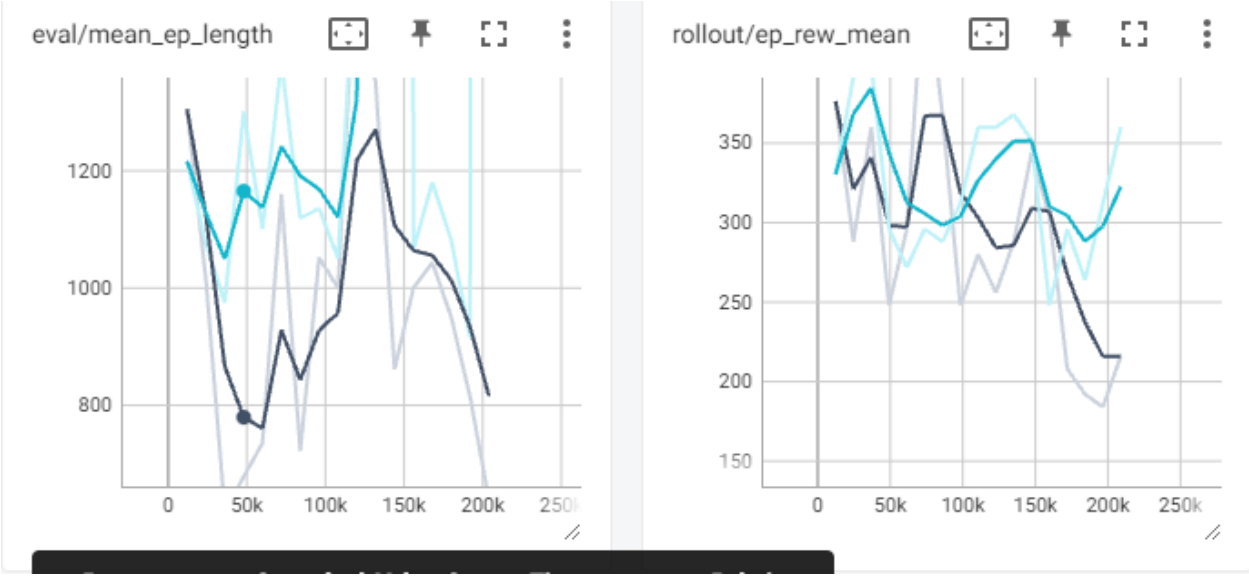

Figure 26 eval/train rewards

Με βάση τα παραπάνω διαγράμματα φαίνεται πως η εκπαίδευση του PPO με Policy CNN default αποδίδει καλύτερα σε σχέση με το Custom CNN που προστέθηκε ως argument. Σ όλες τις μετρικές παρατηρούμε ότι η μπλε γραμμή που αντιστοιχεί στο PPO με Policy CNN default υπερτερεί της σκούρης μπλε γραμμής που αντιστοιχεί στο Custom CNN. Αυτό κατά πάσα πιθανότητα οφείλεται στην αρχιτεκτονική του δικτύου που έχει επιλεχθεί. Η διάρκεια ζωής του agent είναι μεγαλύτερη και τα rewards τα οποία λαμβάνει είναι εξίσου μεγαλύτερα όπως επίσης και τα losses φαίνονται ήδη από την αρχή πιο ισορροπημένα.