**Yazılım Gereksinimleri:**

**Tensorflow env için:**

* Python 3.9.7
* Cuda 11.2
* CudNN: 8.1.1
* Tensorflow 2.6.0

Buradan aşağı için yazdığım versiyonlar yüklenmiş olan versiyonlar büyük ihtimalle son versiyonlarıda çalışacaktır.

* Pip 22.2.2
* Matplotlib 3.4.3
* Keras-tuner 1.0.4
* Jupyterlab 3.1.14
* Pandas 1.3.3
* Shap 0.40.0
* Seaborn 0.11.2
* Scikit-learn 1.0

**Pytorch Env için:**

* Cuda 11.2
* Python 3.9.7
* CudNN: 8.1.1
* Torch 1.11.0+cu113
* Tsai 0.3.1 :https://github.com/timeseriesAI/tsai
* Optuna 2.10.0

Yazılım hem PyTorch hem Tensorflow Framework’ü kullanabilmektedir.

Tensorflow framework’ünde çalışmak için PyCharm IDE’si içinde sağ alt köşede yer alan interpreter setting kısmından python 3.9(env)’nı seçin ve main.py dosyasını çalıştırın



Pytorch İçin Python 3.9 (torch) seçin ve main\_torch.py dosyasını çalıştırın



PyTorch tarafında tensorflow’a göre daha fazla model implement edilmiş durumda. PyTorch tarafında implement edilmiş modelleri Tensorflow tarafında kullanmak istiyorsanız, models dosyasının altında fcn.py’a benzer bir kod yazmanız gerekmektedir.

**Tensorflow ve main.py:**

Ilk olarak import edilen dosyalara bakalım.

CROSS\_VAL\_SPLIT, PREPARATION ve ihtar\_utils

**CROSS\_VAL\_SPLIT Class**: Verilerin belirli bir kurala dayanarak train ve validasyon datası olarak ayrıştırılması.

Pozitif veriler için step\_size parametresi önemli bir parametredir. Drone uçuşlarını bir bütün olarak düşünüp bir csv dosyası içerisinde yer alan pozitif sample sayısını step size değerine bölüp, train ve validasyon oranlarına göre veri setlerine dahil ediliyor.

Bir örnek üzerinden gösterelim:

Örneğin 1000 sample’lık bir pozitif verimiz olsun. Step\_size=2, train\_ratio=0.8, val\_ratio=0.2 olsun.

Yukarıdaki durum için 1000 sample’lık verinin step\_size 2 olduğu için 500-500 düşünebiliriz.

İlk 500 verinin 500\*0.8=400 lik kısmı traine , geri kalan 100 lük kısmı ise validasyona gidiyor.

2. 500’lük veri için aynı şeyi uyguluyoruz. 500\*0.8 = 400’lük kısmı traine, geri kalan 100 lük kısmı validasyona gidiyor.

Bunun amacı validasyon datasının train datasını model eğitim sırasında doğru bir şekilde valide etmesini istemem. Pozitif cross val split için farklı metodlar denesemde şu ana kadar en işe yarayan metod bu oldu.

Negatif veriler için csv dosyası içerisinde yer alan toplam negatif veri sayısı, train\_ratio ile çarpılır.

Örneğin 1000 negatif sample olsun.

1000\*0.8 =800

Bu 800 veriyi traine atmak için, unique id ler için bir for dönüyoruz. Her bir unique id yi train veri seti 800’e ulaşana kadar train verisetine ekliyoruz. 800’e ulaşınca geri kalan unique idleri validasyon verisetine ekliyoruz.

Negatif cross validasyon split için bu tekniği kullanmamın sebebi negatif veriler birbirinden çok bağımsız cisimlerin sampleları olabilmektedir. Örneğin bir kuş ve arabanın ürettiği iz raporu çok farklı olacaktır. Bu yüzden hem birbirine karışmaması için hemde birbirinden farklı cisimler olduğu için pozitif veriler için kullandığımız tekniği negatif veriler için kullanamıyoruz.

Bu cross val split tekniğini her bir csv dosyası için tekrarlıyoruz. İşlem bitince /lstm/data/train.csv ve val.csv olarak kaydediliyor.

Not: Bu class daha fonksiyonel bir hale getirilip daha kullanışlı bir hale getirilebilir. Bu sayede farklı cross validation split tekniklerini kullanmak daha kolay hale gelecektir.

**PREPARATION classı:**

Preparation class’ında oluşturulan train ve val veri setleri bir sözlük yapısına ekleniyor. Sözlük yapısına eklenmesinin sebebi kullanım kolaylığı oluşturmak. Bu sözlük yapısına daha sonra yapılmak istenen test veri setleri de eklenebilir. Include extra test dataset fonksiyonun altında test dosyasının dizini verilebilir. Trainden standart sapma ve ortalama, test ve validasyona uygulanıyor. Ardından sliding window fonksiyonu ile data çoğaltma işlemi yapılıyor.

**SlidingWindow fonksiyonu:** [**https://machinelearningmastery.com/time-series-forecasting-supervised-learning/#:~:text=The%20use%20of%20prior%20time,or%20size%20of%20the%20lag**](https://machinelearningmastery.com/time-series-forecasting-supervised-learning/#:~:text=The%20use%20of%20prior%20time,or%20size%20of%20the%20lag)**.**

**Ihtar\_utils.py:**

Make\_dir fonksiyonunu kullanarak dosya oluşturur ve diğer faydalı fonksiyonlar buraya eklenebilir.

**Main.py ve Parametreleri:**

**mode**: 0 ve 1 değerini alabilir. Hangi radar verisini kullanmak istediğini belirten parametredir. 0 acar **füzyon**, 1 echodyne

**framework**: 0 ve 1 değerini alabilir. Pytorch kullanacaksak framework 0, tensorflow kullanacaksak 1 olarak assign etmek gerekiyor.

**Step\_size**: cross\_val\_split classında detaylı olarak anlatılmıştır. Önerilen değer 2

**Window\_size**: Model ilk çıktısını kaçıncı sample alınca verecek. Model şeklini belirleyen önemli parametrelerden. (batch\_size, window\_size, feature\_size)

**Stride**: Zaman serisinde pencerenin adım atma büyüklüğü. 24 örneklemi bulunan bir zaman serisinde window\_size:16 ve 4: stride 0-16, 4-20, 8-24 şeklinde pencereler oluşturmasını sağlar. Bu sayede model verinin her yerini farklı perspektifden görmesi sağlanır.

**Shuffle\_size**: Verileri pencerelere alıp array haline getirdikten sonra, zaman yapısını bozmadan karıştırmayı sağlayan parametre. Toplam veri sayısından büyük olması gerekmektedir.

**Batch\_size\_list**: denemek istediğin batch\_size değerlerini listenin içine yazabilirsiniz. [64,128,256…]

**Label\_features**: modelin içine dahil edilmeyen ama veri hazırlarken kullanılan sütunlar

**Architecture:** model mimarisini seçmek için gereken parametre. Default: fcn, Farklı modeler kullanmak için models dosyasının altına oluşturabilir ve yazılıma uygulayabilirsinz.

**Observation\_name**: eğitim sırasında neyi gözlemlediğimize dair dosyanın ismi

**Lr\_list**: denenmek istenilen learning rateleri bu listenin içine yazılabilir. Çok büyük learning rateler önerilmez.

**Loss\_patience\_list**: learning rate list ile doğrudan ilişkili olup, stop early fonksiyonun patience parametresidir. Learning rate büyüdükçe patience değeri küçülmelidir. Learning rate ve patience aynı indexte olmalı.

**Max\_trials**: en iyi modeli bulmak için yapılacak arama(deneme) sayısı.

**Number\_of\_models\_to\_save:** yapılan arama sayısı arasında modeller en iyiden en kötüye doğru sıralanır. Belirlenen değer kadar en iyi model kaydedilir.

**Pos\_train\_dts\_ratio\_l**: 0-1 arasında değer alır. Pozitif verisetinde %kaçı train % kaçı validasyon olacağını belirleyen parametredir. Liste içinde verilecek değerler sırayla farklı kombinasyonlar denenecektir.

**Neg\_train\_dts\_ratio\_l:** 0-1 arasında değer alır. Negatif verisetinde %kaçı train % kaçı validasyon olacağını belirleyen parametredir. Liste içinde verilecek değerler sırayla farklı kombinasyonlar denenecektir.

Örneğin ilk etapta 0.6 pozitif train, 0.4 pozitif validasyon ve 0.6 negatif train, 0.4 negatif validasyon verisi ile veri seti oluşturulacaktır. Bu en optimize veri seti oranını bulmak için yapılır.

Main.py çalıştırıldığı zaman saved\_models dizinin altında seçilen architecture ismiyle başlayan bir klasör oluşturulacaktır. O dosyanın altında da çalıştırılan tarih karşılığı ile bir dosya daha oluşturulur. Tarih dosyasının altında da batch\_size\_dataset\_ratio\_learning\_rate ismiyle bir klasör oluşur. Bu klasörün altında da data, models, plots ve test\_results.csv dosyaları oluşur.

Models klasörün altında eğitilen modeller kaydedilir. Numbers\_of\_models\_to\_save kadar model kaydedilir. Loss/acc grafiği ve standard scalerde kaydedilir.

Data klasörün altında model için hazırlanan train ve validasyon verisi yer alır. Df\_config.csv veri hazırlama sırasında istatisteksel bir tablo hazırlar. Bu sayede train’e kaç veri gitmiş val’a kaç veri gitmiş bu verilerin %kaçı pozitif veya negatif bunu görmüş oluruz.

İç içe yer alan 4 for döngüsü yukarıda girilen parametreler ile farklı kombinasyonların denenmesi ve kaydedilmesi sağlanır.

**Train.py**

Model eğitildiği dosyadır.

TRAIN\_FCN classı FCN mimarisini eğitmek için kullanılan sınıftır. Main.py dosyasında belirlenen eğitim parametreleri buraya aktarılır. FCN mimarisinin farklı varyasyonları vardır. MALSTM-FCN, MLSTM-FCN…

**Fcn.py**

Kerastuner’ın search yapısı kullanılmıştır. Model için uygun olabilecek uygun parametre aralıkları verilir. Keras tuner bu parametreleri bayesian optimizasyon algoritması ile aralarından en iyisini bulmaya çalışır.

<https://keras.io/keras_tuner/>

<https://neptune.ai/blog/keras-tuner-tuning-hyperparameters-deep-learning-model>

Tune edilebilecek parametreler: learning rate, optimizer, nöron sayısı, filtre sayısı, kernel size, layer sayısı gibi birçok parametre keras tuner ile optimize edilebilir.

**BayesianOptimization\_FCN** fonksiyonu:

Keras tuneri içinde barındıran model eğitim algoritmasının çalıştığı fonksiyondur.

Tuner.search ile model için en iyi hiperparametreleri aramalara başlar. En iyi modellerin hiperparametrelerini best\_hps değişkenine atar.

Kaydedilen hiperparametreler ile eğitilerek models dosyasının altına sırasıyla kaydedilir.

<https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/EarlyStopping>

<https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/ReduceLROnPlateau>

<https://keras.io/api/keras_tuner/tuners/bayesian/>

Kaydedilen her model test senaryolarında test edilerek skorları csv dosyasına kaydedilir.

Self.dataset\_dict: içinde test verilerini barındıran sözlüktür. ‘senaryo\_ismi’: senaryo\_verisi

Şeklinde veri tutulur. /data/test\_fusion\_dataset/ dizinin altına eklenecek test dosyalarının pathi include\_extra\_test\_dataset fonksiyonun altına diğer örneklerdeki gibi eklenirse otomatikmen test yapısına eklenecektir.

Dosyanın içinde yer alması gereken dosyalar:

extractedData\_All\_Test/Train (öznitelikleri çıkartılmış izraporu)

**Pytorch framework’ünü kullanmak için:**

Main\_torch.py I çalışmak için ilk önce pycharm’da sağ alttan interpreter setting kısmından torch environmentina geçiş yapıyoruz.

main\_torch.py ‘ın main.py dan farklı olan kısmı main\_torch.py ‘ın “tsai” reposunu kullanmasıdır.

Tsai reposu: <https://github.com/timeseriesAI/tsai>

Literatürde yer alan en yeni zaman serisi modelleri repoya implement etmiş ve sürekli güncellenen bir repodur.

Bu repoyu kendi yazılımımıza implement ederek model çeşitlerimizi artırdık.

Tsai\_ihtar klasörü bu repoyu barındırır. Tsai\_ihtar/models klasörürün altındaki modelleri kullanmak için search\_hyper.py dosyasının altındaki yazılan örnek fonksiyonlar gibi search yapacak fonksiyonlar yazmanız gerekiyor. Yazdıktan sonra get\_objective fonksiyonun altına ekleyip main\_torch architecture parametresi ile çağırabilirsiniz.

Search\_hyper.py’da kullanmak istediğiniz modelin tune etmek istediğiniz parametrelerini farklı değer aralıklarla yazıp, arch\_config’in içine ekleyin.

Şu ana kadar implement edilen resnet, rnn\_fcn ve türevleri, inceptiontime ve inceptiontime plus.

Search\_hyper kodunu daha generic bir hale getirip kullanım kolaylığı sağlanabilir.

Veriyi hazırlama kısmı main.py ile aynı. Sadece tensorflow model shapei (data\_size, window\_size, feature\_size)

Pytorchta ise (data\_size, feature\_size, window\_size) şeklinde olmasıdır. main\_torch u çalıştırırken framework 0 yapmayı unutmayın.

My\_learner.py tsai reposunu kullanmak için oluşturulan tsai/learner.py dan esinlenerek oluşturulan bir kodtur.

My\_learner oluşturulma sebebi loss\_acc grafiklerini , modeli istenilen dosya formatında ve dizinine kaydetmek

**Veri Temizleme:**

Workspace klasörünün altında yer alan veri\_temizleme.ipynb dosyasını jupyter-notebook kullanarak istenmeyen verileri atabilirsiniz.

Fonksiyon isimlerinden anlaşılabileceği gibi farklı caseler farklı slime fonksiyonları var.

Örneğin verinin sadece poziitf veya negatif kısmını alınabilir.

Del\_inside\_rect: silmek istediğiniz dikdörtgen içindeki kısmın x ve y kordinatları vermeniz gerekmektedir.

Del\_based\_on\_id: silmek istediğiniz bir id varsa numarasını yazmak yeterli

Del\_ids\_in\_rect: belirtilen dikdörtgen içinde bulunan tüm idlerin verilerini siler. ID eğer dikdörtgenin dışına taşmışsa yine siler.

Del\_partial\_id: idnin bozuk, istenilmeyen kısmını silmek için kullanılır.

Del\_based\_on\_xy: radarın menzili dışında kalan kısımları temizlemek için kullanılır.

Del\_jumped\_data: atlayan verileri silmek için kullanılır. Atlayan verileri bulmak için feature\_engineering altındaki match\_ids kodunu kullanabilirsiniz.

**Öznitelik Analizi**

Öznitelik analizinin yapıldığı kodtur. Yoğunluk grafiği, histogram ve keman grafiğini bastırıp analiz yapabilirsinz.

Plot\_all(‘senaryo\_ismi’) şeklinde tüm grafiklerini bastırabilirsiniz.

Feature Extraction: