# Знакомство с TSAI

Одним из наиболее современных пакетов работы с временными рядами при помощи глубокого обучения нейронных сетей является пакет [**TSAI**](https://github.com/timeseriesAI/tsai), официальная документация по данному пакету может быть найдена [тут](https://timeseriesai.github.io/tsai/).

Пакет tsai — это пакет глубокого обучения с открытым исходным кодом, созданный на основе фреймворков Pytorch и fastai, ориентированный на современные методы для задач временных рядов, таких как классификация, регрессия, прогнозирование, вменение.

Для установки пакета разработчиками рекомендуется следующий скрипт:

stable = True # Set to True for latest pip version or False for main branch in GitHub  
!pip install {"tsai -U" if stable else "git+https://github.com/timeseriesAI/tsai.git"} >> /dev/null

Теперь рассмотрим импорт пакета.

# stable = True # Set to True for latest pip version or False for main branch in GitHub  
# !pip install {"tsai -U" if stable else "git+https://github.com/timeseriesAI/tsai.git"} >> /dev/null

from tsai.all import \*  
import warnings  
  
computer\_setup()

os : macOS-10.16-x86\_64-i386-64bit  
python : 3.9.7  
tsai : 0.3.1  
fastai : 2.7.6  
fastcore : 1.4.5  
torch : 1.10.1  
device : cpu  
cpu cores : 8  
RAM : 16.0 GB  
GPU memory : N/A

from sklearn.model\_selection import train\_test\_split

import torch  
from torch import nn

Теперь давайте выберем данные. Мы будем использовать тот же пример, что и для классификации в SKTime - набором данных [открытые данные энергетических систем](https://open-power-system-data.org/).

def extract\_country(df\_all, country\_code, year\_min=None, year\_max=None):  
 # List of columns to extract  
 columns = [col for col in df\_all.columns if col.startswith(country\_code)]  
 # Extract columns and remove country codes from column labels  
 columns\_map = {col : col[3:] for col in columns}  
 df\_out = df\_all[columns].rename(columns=columns\_map)  
 # Exclude years outside of specified range, if any  
 if year\_min is not None:  
 df\_out = df\_out[df\_out.index.year >= year\_min]  
 if year\_max is not None:  
 df\_out = df\_out[df\_out.index.year <= year\_max]  
 return df\_out  
  
def transform\_dataframe(df, cols\_map):  
 # Rename columns for convenience  
 df = df[list(cols\_map.keys())].rename(columns=cols\_map)   
 df = df / 1000 # Convert from MW to GW  
 df = df.rename\_axis('Date')  
 return df  
  
def split\_by\_month(df, months):  
 df\_ = pd.DataFrame()  
 for month in np.atleast\_1d(months):  
 df\_ = df\_.append(df.loc[df.index.month == month])  
 return df\_  
  
def to\_segments(df, column, size = 24):   
 df.index.hour[0]  
 start\_idx = 24-df.index.hour[0]  
 df = df.iloc[start\_idx:]  
 val = df[[column]].values  
 return val[:size\*(val.size//size)].reshape(-1,size)

url = 'https://data.open-power-system-data.org/time\_series/2020-10-06/'  
datafile = url + 'time\_series\_60min\_singleindex.csv'  
df\_all = pd.read\_csv(datafile, index\_col='utc\_timestamp', parse\_dates=True, low\_memory=False)  
  
df\_hrly = extract\_country(df\_all, country\_code='DE', year\_min=2015, year\_max=2019)  
  
df\_hrly.sample(3)  
  
cols\_map = {'load\_actual\_entsoe\_transparency' : 'Consumption',  
 'wind\_generation\_actual' : 'Wind',  
 'solar\_generation\_actual' : 'Solar'}  
df = transform\_dataframe(df\_hrly, cols\_map).dropna()  
  
# df wind + solar generation  
df['Wind+Solar'] = df[['Wind', 'Solar']].sum(axis=1, skipna=False)  
df.to\_csv('de\_clf\_data.csv')  
df.head()

Consumption Wind Solar Wind+Solar  
Date   
2015-01-01 07:00:00+00:00 41.133 10.208 0.071 10.279  
2015-01-01 08:00:00+00:00 42.963 10.029 0.773 10.802  
2015-01-01 09:00:00+00:00 45.088 10.550 2.117 12.667  
2015-01-01 10:00:00+00:00 47.013 11.390 3.364 14.754  
2015-01-01 11:00:00+00:00 48.159 12.103 4.198 16.301

winter = to\_segments(split\_by\_month(df, [1,2, 12]), 'Consumption', size = 24)  
summer = to\_segments(split\_by\_month(df, [6, 7, 8]), 'Consumption', size = 24)  
autumn = to\_segments(split\_by\_month(df, [9,10,11]), 'Consumption', size = 24)  
spring = to\_segments(split\_by\_month(df, [3, 4, 5]), 'Consumption', size = 24)  
print(winter.shape, summer.shape, autumn.shape, spring.shape)  
  
X = np.concatenate((  
 winter,   
 summer,   
 autumn,   
 spring))  
y = np.concatenate((  
 1\*np.ones(winter.shape[0]),  
 2\*np.ones(summer.shape[0]),  
 3\*np.ones(autumn.shape[0]),  
 4\*np.ones(spring.shape[0])  
 ))  
  
  
X = np.atleast\_3d(X).transpose(0,2,1)  
y.astype(int)

(448, 24) (458, 24) (452, 24) (458, 24)

array([1, 1, 1, ..., 4, 4, 4])

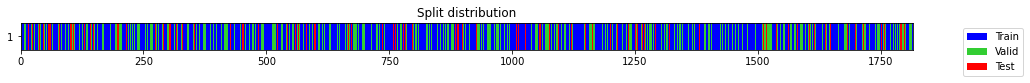
Также давайте сопоставим названия классов и их номера для большего удобства.

class\_map = {  
 1:'winter',   
 2:'summer',   
 3:'autumn',   
 4:'spring',   
 }  
class\_map  
  
labeler = ReLabeler(class\_map)  
y = labeler(y)

Особенностью данного фреймворка (как и его основы fastai) является использование супер высокоуровневых API. Это делает фреймворк tsai практически декларативным. Для проведения экспериментом нам потребуется лишь выбор модели и набора данных. Все обучение будет происходить в backend.

Для работы с наборами данных давайте создадим их разделение на тренировочную, валидационную и тестовую подвыборки. Для этого воспользуемся функцией get\_splits. В рамках урока мы задаем аргумент show\_plot чтобы визуализировать разделение данных.

splits = get\_splits(y,   
 n\_splits=1,   
 valid\_size=0.3,   
 test\_size=0.1,   
 shuffle=True,   
 balance=False,   
 stratify=True,  
 random\_state=42,   
 show\_plot=True,   
 verbose=True)  
splits



((#1091) [1274,358,488,1233,469,725,151,1202,1173,894...],  
 (#544) [1783,1159,952,1712,203,45,563,1071,1640,868...],  
 (#181) [690,310,767,645,80,673,451,597,927,1404...])

Для начала давайте создадим набор данных. Такой набор будет экземпляром класса TSDatasets. В наборе данных зададим разделение данных и необходимые преобразования tfms.

Также сконфигурируем загрузчик батчей TSDataLoaders. Загрузим тренировочный и валидационный наборы данных.

tfms = [None, [Categorize()]]  
dsets = TSDatasets(X, y, tfms=tfms, splits=splits)  
   
bs = 256  
dls = TSDataLoaders.from\_dsets(dsets.train, dsets.valid, bs=[bs, bs\*2])

Среди анализируемых архитектур мы рассмотрим рекуррентные RNN, LSTM и GRU ячейки в двух вариантах обычном и плюс. Варианты несколько отличаются по архитектуре. Для обоих вариантов мы рассмотрим архитектуры с 3 и 4 слоями, как двунаправленные, так и нет.

Кроме рекуррентных нейронных сетей мы также попробуем ряд стандартных сверточных архитектур, таких как ResNet1d, xresnet1d, InceptionTime, XceptionTime и TCN. Данные архитектуры представляют собой стандартные подходы к классификации временных рядов нейронными сетями. Последняя архитектура представляет собой использование идеи расширенной свертки. архитектуры ResNet1d и xresnet1d отличаются подходом к организации skipconnection. Архитектуры InceptionTime и XceptionTime представляют собой адаптированные под временные ряды соответствующих сверточных сетей для изображений. Также, для сравнения, мы попробуем использовать смешанную архитектуру LSTM\_FCN, полносвязную архитектуру FCN и архитектуру трансформер TST.

Отметим, что почти все архитектуры в нашем примере мы вызываем "из коробки", подробней с реализациями данных архитектур можно ознакомится в рамках официальной документации.

archs = [  
 (RNNPlus, {'n\_layers':3, 'bidirectional': True} ),  
 (LSTMPlus,{'n\_layers':3, 'bidirectional': True} ),  
 (GRUPlus, {'n\_layers':3, 'bidirectional': True} ),   
 (RNNPlus, {'n\_layers':4, 'bidirectional': True} ),  
 (RNNPlus, {'n\_layers':4, 'bidirectional': False}),   
 (LSTM, {'n\_layers':3, 'bidirectional': False}),   
 (RNN, {'n\_layers':3, 'bidirectional': True} ),   
 (LSTM, {'n\_layers':3, 'bidirectional': True} ),  
 (GRU, {'n\_layers':3, 'bidirectional': True} ),   
 (ResNet, {}),   
 (xresnet1d34, {}),   
 (xresnet1d50\_deeper, {}),   
 (InceptionTime, {}),   
 (XceptionTime, {}),   
 (TCN, {}),   
 (LSTM\_FCN, {}),   
 (TST, {}),  
 (FCN, {}),   
 ]

Теперь попробуем запустить обучение. Для создания модели используем функцию create\_model, которая для создает модель в формате совместимом с PyTorch для заданной архитектуры и набора гиперпараметров.

В пакете tsai обучение проводится для объекта класса Learner. Таким образом этот класс описывает модель (обученную архитектуру). Мы приведем лишь самую простую конфигурацию этого класса. Конфигурация включает указание загрузчика данных, модели и метрики. В качестве метрики мы используем точность (accuracy). Для запуска обучения в классе Learner используется метод fit\_one\_cycle. Для этого метода в качестве параметров мы зададим число эпох обучения и скорость обучения. Остальные параметры оставим по умолчанию. Среди таких параметров будут функция потерь (кросс энтропия) и метод оптимизации ADAM.

По результатам обучения каждой модели мы занесем в таблицу полученные результаты, в том числе точность для валидационного набора данных и время обучения.

from IPython.display import clear\_output  
  
  
  
results = pd.DataFrame(columns=['arch', 'hyperparams', 'total params', 'train loss', 'valid loss', 'accuracy', 'time'])  
  
for i, (arch, k) in enumerate(archs):  
   
 model = create\_model(arch, dls=dls, \*\*k)  
   
 print(model.\_\_class\_\_.\_\_name\_\_)  
   
 learn = Learner(dls, model, metrics=accuracy)  
 start = time.time()  
 learn.fit\_one\_cycle(20, 1e-3)  
 elapsed = time.time() - start  
 vals = learn.recorder.values[-1]  
 results.loc[i] = [arch.\_\_name\_\_, k, count\_parameters(model), vals[0], vals[1], vals[2], int(elapsed)]  
 results.sort\_values(by='accuracy', ascending=False, ignore\_index=True, inplace=True)  
 clear\_output()  
 display(results)

arch hyperparams total params \  
0 ResNet {} 478724   
1 xresnet1d34 {} 7229284   
2 InceptionTime {} 388868   
3 xresnet1d50\_deeper {} 21224548   
4 XceptionTime {} 399480   
5 FCN {} 264580   
6 LSTM\_FCN {} 315380   
7 TST {} 411524   
8 RNNPlus {'n\_layers': 4, 'bidirectional': False} 71304   
9 RNNPlus {'n\_layers': 4, 'bidirectional': True} 202604   
10 RNNPlus {'n\_layers': 3, 'bidirectional': True} 142204   
11 RNN {'n\_layers': 3, 'bidirectional': True} 142204   
12 GRU {'n\_layers': 3, 'bidirectional': True} 425004   
13 GRUPlus {'n\_layers': 3, 'bidirectional': True} 425004   
14 LSTMPlus {'n\_layers': 3, 'bidirectional': True} 566404   
15 LSTM {'n\_layers': 3, 'bidirectional': True} 566404   
16 LSTM {'n\_layers': 3, 'bidirectional': False} 203204   
17 TCN {} 66754   
  
 train loss valid loss accuracy time   
0 0.274329 0.150533 0.950368 51

Среди полученных результатов хотелось бы выделить RNNPlus. Эта архитектура дала результат, оптимальный по соотношению точности и времени работы. Давайте рассмотрим эту архитектуру подробней

arch = RNNPlus  
k = {'n\_layers': 4, 'bidirectional': True}  
model = create\_model(arch, dls=dls, \*\*k)

model = build\_ts\_model(RNNPlus, arch\_config=k, dls=dls)

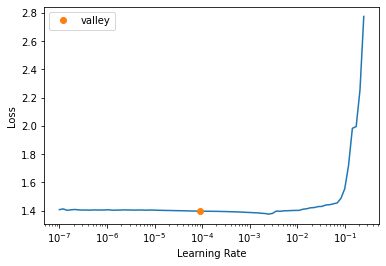
Фреймворк tsai предлагает метод определения оптимальной скорости обучения на основе тестов нескольких эпох обучения. Попробуем определить лучшую скорость обучения.

learn = ts\_learner(dls, arch=model, metrics=accuracy)   
learn.lr\_find()

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

SuggestedLRs(valley=9.120108734350652e-05)

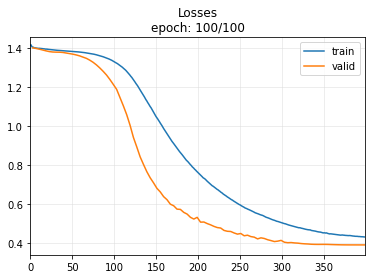


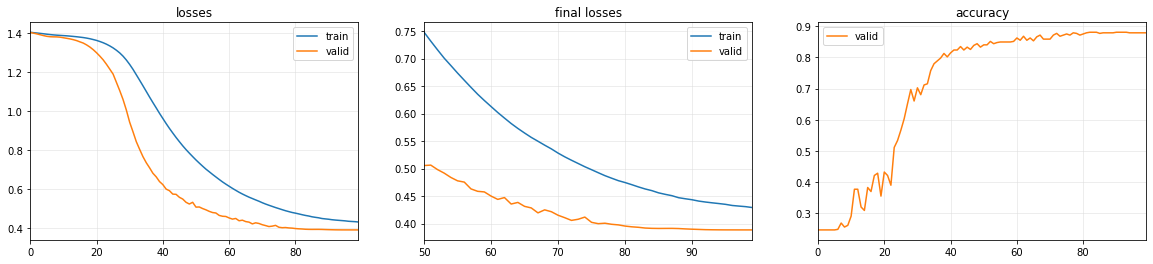
Теперь снова запустим обучение. В этот раз мы запустим процедуру тренировки с коллбеком, позволяющим отслеживать значения функций потерь в режиме он-лайн.

learn = Learner(dls, model, metrics=accuracy)  
start = time.time()  
learn.fit\_one\_cycle(n\_epoch = 100, lr\_max = 0.00009, cbs=ShowGraph())

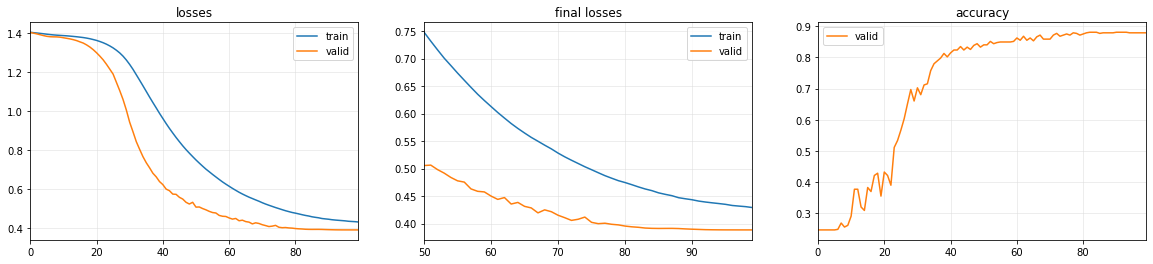
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>





learn.plot\_metrics()

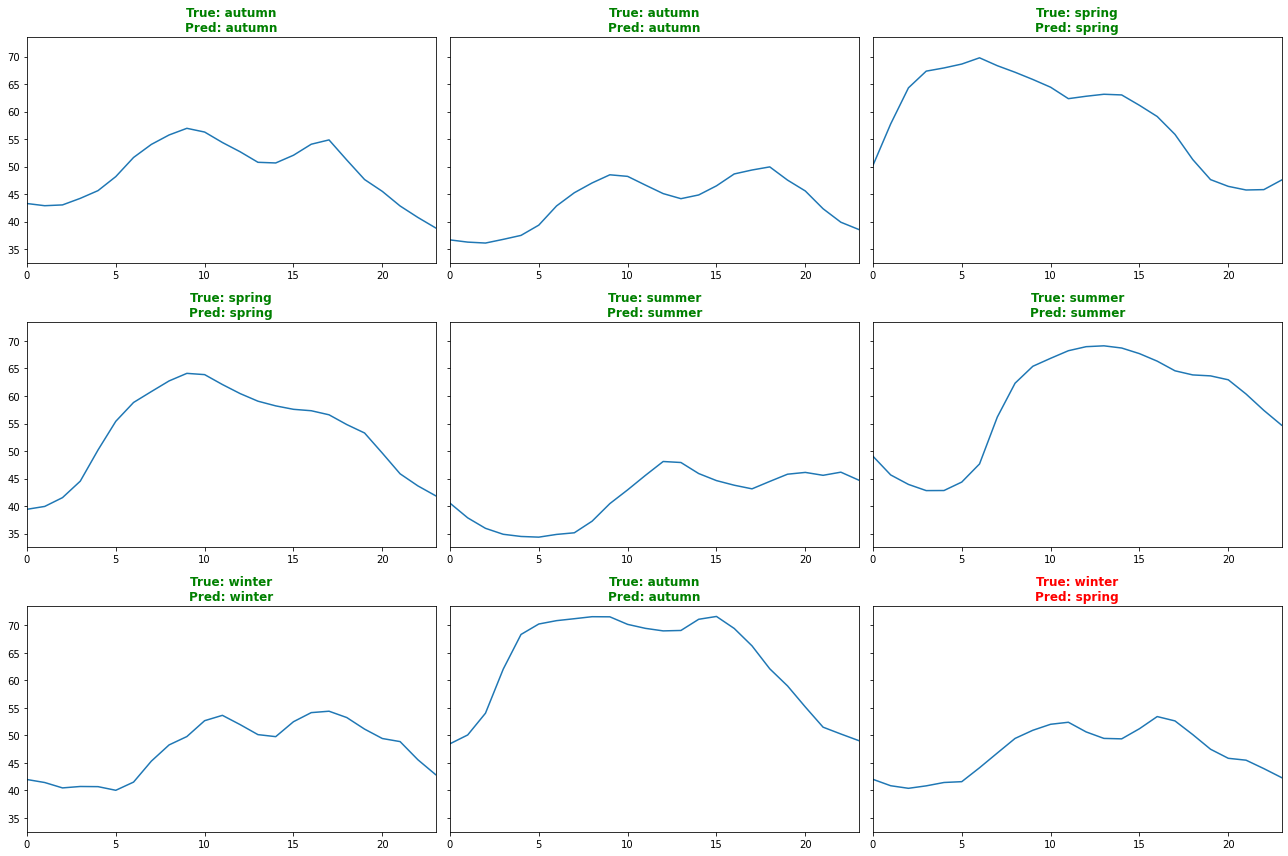


Рассмотрим несколько примеров результатов для валидационного набора данных.

learn.show\_results(sharey=True)

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

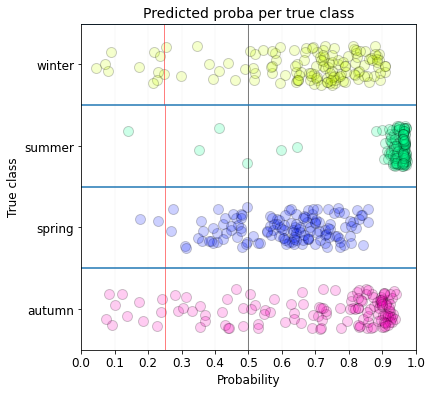


Также давайте попробуем визуализировать результаты. Следующий график показывает вероятность при которой определяется класс для каждого семпла валидационной выборки.

learn.show\_probas()

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>



Также для анализа ошибок давайте посмотрим на матрицу ошибок.

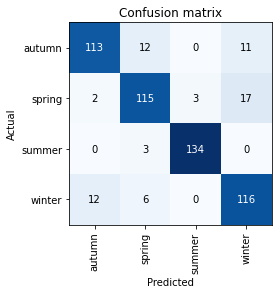
interp = ClassificationInterpretation.from\_learner(learn)  
interp.plot\_confusion\_matrix()

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>



Выберем из матрицы наиболее ошибочные случаи.

interp.most\_confused(min\_val=3)

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

[('spring', 'winter', 17),  
 ('autumn', 'spring', 12),  
 ('winter', 'autumn', 12),  
 ('autumn', 'winter', 11),  
 ('winter', 'spring', 6),  
 ('spring', 'summer', 3),  
 ('summer', 'spring', 3)]

Помимо прочего попробуем сделать предсказания для тестового набора данных.

probas, \_, preds = learn.get\_X\_preds(X[splits[2][:5]])  
preds, y[splits[2][:5]]

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

("['summer', 'winter', 'summer', 'summer', 'winter']",  
 array(['summer', 'winter', 'summer', 'summer', 'winter'], dtype='<U6'))

Оценим точность на тестовом наборе данных

probas, \_, preds = learn.get\_X\_preds(X[splits[2]])

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

class\_map = {  
 'winter':3,   
 'summer':2,   
 'autumn':0,   
 'spring':1,   
 }  
class\_map  
  
labeler = ReLabeler(class\_map)

from sklearn.metrics import accuracy\_score  
  
accuracy\_score(np.argmax(probas,axis=-1), labeler(y[splits[2]]))

0.8674033149171271

также покажем возможность сохранить и загрузить наборы данных при помощи tsai.

learn.save\_all(path='export', dls\_fname='dls', model\_fname='model', learner\_fname='learner')

learn = load\_learner\_all(path='export', dls\_fname='dls', model\_fname='model', learner\_fname='learner')  
dls = learn.dls

Отдельно выгрузить и загрузить модель для работы (не для тренировки) можно при помощи следующего скрипта.

PATH = Path('./models/RNNn.pkl')  
PATH.parent.mkdir(parents=True, exist\_ok=True)  
learn.export(PATH)

learn = load\_learner(PATH)

probas, \_, preds = learn.get\_X\_preds(X[splits[2][:5]])  
preds, y[splits[2][:5]]

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

("['summer', 'winter', 'summer', 'summer', 'winter']",  
 array(['summer', 'winter', 'summer', 'summer', 'winter'], dtype='<U6'))

Больше информации по работе с фреймворком tsai можно получить в разделе уроки в рамках [официального репозитория](https://github.com/timeseriesAI/tsai/tree/main/tutorial_nbs) или [документации](https://timeseriesai.github.io/tsai/tutorials.html).

1 Какая функция в пакете TSAI позволяет разделить данные

Ответ: get\_splits

2 Какая часть полученного разделения данных split соответствует валидационным данным (ответ числом)

Ответ : 1

3 Напишите название рекуррентной ячейки, имеющей два выхода для сохранения долговременного контекста и кратковременного контекста

Ответ: LSTM

4 Выберете ответ, зачем нужна двунаправленная рекуррентная ячейка:

* Возможность увеличить глубину сети
* Снижение влияния проблемы усеченного обучения
* Казуальность
* Проблема гетеросекдастичности

Ответ Снижение влияния проблемы усеченного обучения

5 Выберите причину сложности обучения RNN ячеек

* Переиспользование одних и тех же весовых параметров
* Большая глубина сети
* Усеченое обучение
* Большой объем параметров

Ответ: Переиспользование одних и тех же весовых параметров

6 Сопоставьте классы архитектур

* InceptionTime - сверточная сеть
* GRU - рекуррентная сеть
* xresnet - сверточная сеть
* RNNPlus - рекуррентная сеть
* FCN – полносвязная сеть

7 Для какой архитектуры мы получили самые точные результаты:

Ответ: ResNet

8 Для какой архитектуры мы получили самое быстрое время работы:

Ответ:

|  |  |
| --- | --- |
| RNNPlus | {'n\_layers': 4, 'bidirectional': False} |

Или RNNPlus

9 Сколько слоев было в архитектуре, которую мы выбрали по результатам отбора

Ответ: 4

10 При помощи какого метода мы провели предварительную оценку скоротси обучения

Ответ: learn.lr\_find() или lr\_find() или lr\_find или learn.lr\_find

11 При помощи какого метода мы провели обучение сети

Ответ: fit\_one\_cycle Или learn.fit\_one\_cycle или fit\_one\_cycle() Или learn.fit\_one\_cycle()

12 Для каких данных мы получаем предсказания для следующей записи learn.get\_X\_preds(X[splits[2]])

* Тренировочные
* Валидационные
* Тестовые
* Все вместе

Ответ: Тестовые

13 Как интерпретировать результат функции interp.most\_confused(min\_val=3)

* Наиболее ошибочные пары результатов
* Классы с большим числом правильных ответов
* Классы с наибольшим числом семплов
* Классы с наименьшим числом семплов

Ответ: Наиболее ошибочные пары результатов