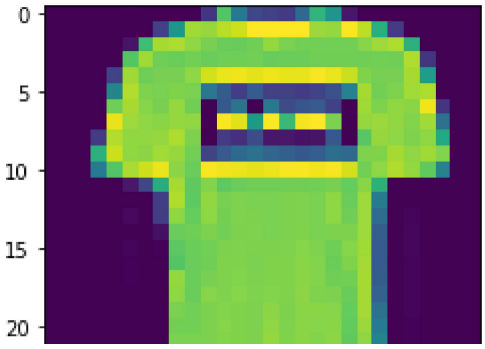
```
# Код нейронной сети в Keras!
################################
import numpy as np
import pandas as pd

#import mnist
from keras.models import Sequential
from keras.layers import Dense
from keras.datasets import mnist, fashion_mnist
from tensorflow.keras.utils import to_categorical


(train_imagesi,train_labels),(test_imagesi,test_labels)=fashion_mnist.load_data()

# можно брать mnist из специальной библиотеки
# train_images = fashion_mnist.train_images()
#train_labels = fashion_mnist.train_labels()
#test_images = fashion_mnist.test_images()
#test_labels = fashion_mnist.test_labels()

# print(train_images.shape)
# приведем изображение к диапазону (-1,1).
train_images = (train_imagesi / 127) - 1
test_images = (test_imagesi / 127) - 1

# делаем векторизацию, т.к. наши слои полносвязные и хотят на вход вектор.
train_images = train_images.reshape((-1, 784))
test_images = test_images.reshape((-1, 784))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/tra
32768/29515 [================================] - 0s 0us/step
40960/29515 [========================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/tra
26427392/26421880 [==============================] - 0s 0us/step
26435584/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10
16384/5148 [================================================================================
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10
4423680/4422102 [==============================] - 0s 0us/step
4431872/4422102 [==============================] - 0s 0us/step
```

```
import matplotlib.pyplot as plt


plt.imshow(train_imagesi[1,:,:])
plt.show()
print(train_labels[2])
```



```
np.unique(train_labels)
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=uint8)
```

```
0
```

Создание Модели

```
# res = pd.DataFrame(columns=('name','epochs','n','optimizer', 'hid_layer', 'train_accurac
# res
```

name	epochs	n	optimizer	hid_layer	train_accuracy	test_accuracy

```
y_t = to_categorical( train_labels)
y_tt = to_categorical(test_labels)


lay = np.arange(1, 11)
# N = np.arange(10, 101, 10)
# N = [10, 50 , 100]
N = [10]
# EPOCHS = [5,15]
EPOCHS = [5]
# N = [1]
for k in N:
  model1 = Sequential(name='10_lay_my_model'+ f'_{k}')
  model1.add(Dense( 128, activation='relu', input_shape=(784,)))
  # входной слой + поносвязный слой из 128 нейронов с активацией ReLU

  model1.add(Dense(k, activation='tanh'))
  model1.add(Dense(k, activation='tanh'))
  model1.add(Dense(k, activation='tanh'))
  model1.add(Dense(k, activation='tanh'))
  model1.add(Dense(k, activation='tanh'))
  model1.add(Dense(k, activation='tanh'))
  model1.add(Dense(k, activation='tanh'))
  model1.add(Dense(k, activation='tanh'))
  model1.add(Dense(k, activation='tanh'))
  model1.add(Dense(k, activation='tanh'))
  # Скрытый слой k из 10+(10-k)*10 нейронов с активацией tanh

  model1.add(Dense(10, activation='softmax'))
  # выходной слой из 10 нейронов (сколько классов, столько нейронов) + активация softmax
```

```
# model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# for i_optim in ['SGD','RMSProp','adam','NAdam']:
for i_optim in ['adam']:
  model1.compile(
    optimizer= i_optim, # оптимизатор
    loss='binary_crossentropy', # функция потерь
    metrics=['accuracy'], # метрика
  )
  # Train the model.
  for ep in EPOCHS:
    model1.fit(
      train_images,

      y_t,
      epochs= ep,
      batch_size=32, validation_split=0.2
    )

    # Evaluate the model.
    model1.evaluate(
      test_images,
      y_tt
    )
    res.loc[len(res.index)] = [model1.name,
                               ep,
                               k,
                              i_optim,
                              10,
                              round(model1.evaluate(train_images,y_t)[1],4),
                              round(model1.evaluate(test_images, y_tt)[1],4) ]
    print(res)
```

```
Epoch 1/5
1500/1500 [==============================] - 7s 4ms/step - loss: 0.2752 - accuracy: 0
Epoch 2/5
1500/1500 [==============================] - 6s 4ms/step - loss: 0.1784 - accuracy: 0
Epoch 3/5
1500/1500 [==============================] - 6s 4ms/step - loss: 0.1648 - accuracy: 0
Epoch 4/5
1500/1500 [==============================] - 6s 4ms/step - loss: 0.1608 - accuracy: 0
Epoch 5/5
1500/1500 [==============================] - 6s 4ms/step - loss: 0.1550 - accuracy: 0
313/313 [==============================] - 1s 2ms/step - loss: 0.1522 - accuracy: 0.0
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1504 - accuracy: 0
313/313 [==============================] - 1s 2ms/step - loss: 0.1522 - accuracy: 0.0
                  name epochs   n  ... hid_layer train_accuracy  test_accuracy
0  10_lay_my_model_10      5  10  ...        10         0.6273         0.6244

[1 rows x 7 columns]
```

```
res
```

```
# res.to_csv('result.csv')
from google.colab import files
res_csv = files.upload()
```

Выбрать файлы result.csv

- **result.csv**(application/vnd.ms-excel) - 4843 bytes, last modified: 06.12.2021 - 100% done

```
Saving result.csv to result.csv
```

```
import pandas as pd
res = pd.read_csv('result.csv', index_col=0)
```

```
res.head(-5)
```

	name	epochs	n	optimizer	hid_layer	train_accuracy	test_acc
0	1_lay_my_model_10	5	10	SGD	1	0.6919	0
1	1_lay_my_model_10	15	10	SGD	1	0.8267	0
2	1_lay_my_model_10	5	10	RMSProp	1	0.8753	0
3	1_lay_my_model_10	15	10	RMSProp	1	0.9078	0
4	1_lay_my_model_10	5	10	adam	1	0.9130	0
...	...	...	...	...	...	...	
86	10_lay_my_model_50	5	50	NAdam	10	0.8999	0
87	10_lay_my_model_50	15	50	NAdam	10	0.9049	0
88	10_lay_my_model_100	5	100	SGD	10	0.8057	0
89	10_lay_my_model_100	15	100	SGD	10	0.8808	0
90	10_lay_my_model_100	5	100	RMSProp	10	0.8600	0

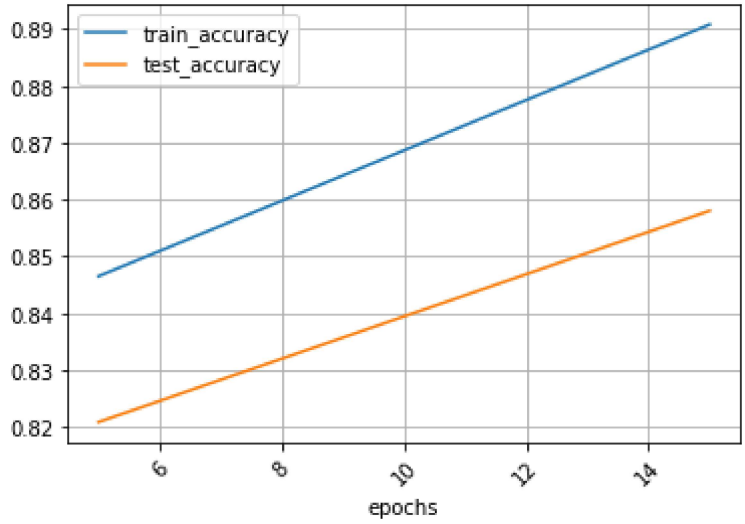
91 rows × 7 columns

```
res.groupby('n')[['train_accuracy', 'test_accuracy']].mean().plot(grid=True, rot= 45)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6b43b2f390>
```

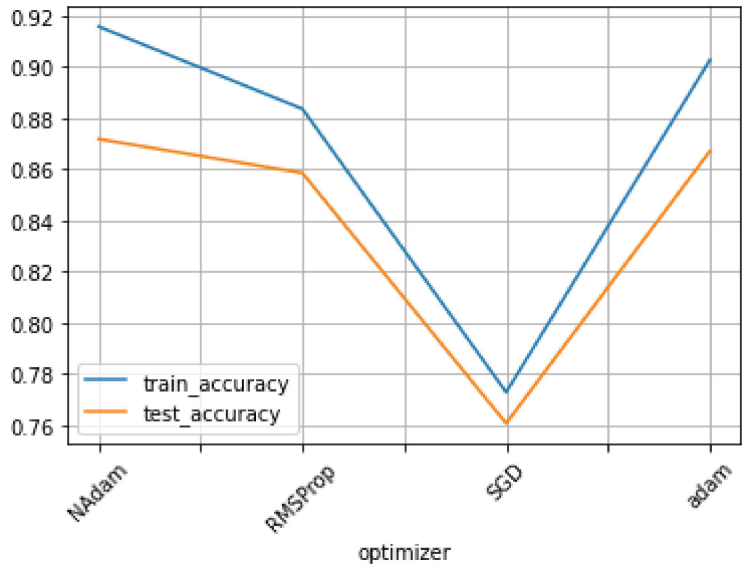
```
res.groupby('epochs')[['train_accuracy', 'test_accuracy']].mean().plot(grid=True, rot= 45)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6b43ba55d0>
```



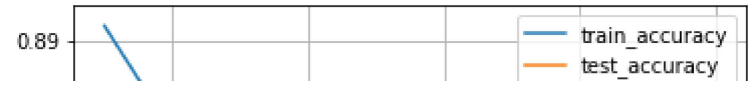

```
res.groupby('optimizer')[['train_accuracy', 'test_accuracy']].mean().plot(grid=True, rot=
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6b43c91b90>
```



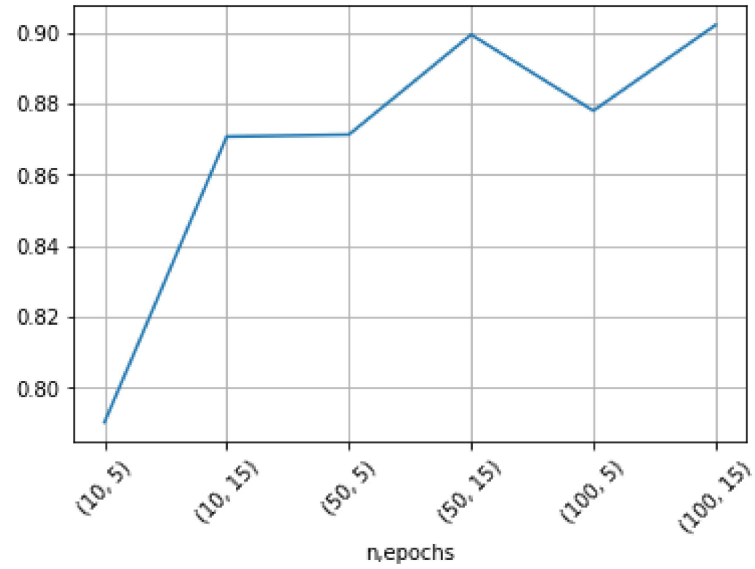

```
res.groupby('hid_layer')[['train_accuracy', 'test_accuracy']].mean().plot(grid=True, rot=
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6b43d52610>
```




```
res.groupby(['n', 'epochs'])['train_accuracy'].mean().plot(grid=True, rot= 45)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6b439ba690>
```



По графикам можно увидеть, что нарезультат влияют (n)- кол-во нейронов в слое, чем больше тем результат выше
(epochs) кол-во итераций по данным, получилась линейная зависимость , чем больше тем лучше
среди оптимизаторов(optimizer) лучше всего себя показал Adam и NAdam, на втором месте RMSProp , третье место по графику и совсем не самый лучший это SGD
почему то кол-во слоев(hid_layer) чем больше тем хуже результат на тесте, хотя по таблице видно, что если слоев больше, то точность, после первого обучения сети сразу выше, возможно для хорошего результа нужно меньше итераций

```
pred = model1.predict(test_images)
```

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

```
y_pred = np.argmax(model1.predict(test_images), axis=1)
y_pred[:10]
```

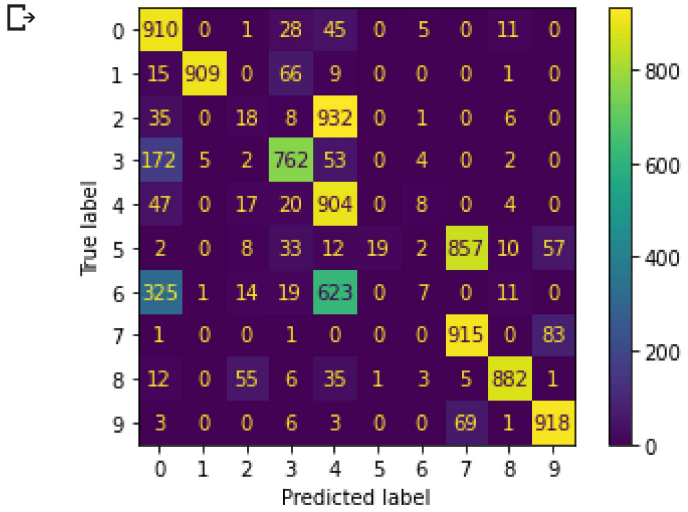
```
array([9, 4, 1, 1, 4, 1, 4, 4, 7, 7])
```

```
CM = confusion_matrix(test_labels,y_pred)
print(CM)
```

```
[[910   0   1  28  45   0   5   0  11   0]
 [ 15 909   0  66   9   0   0   0   1   0]
 [ 35   0  18   8 932   0   1   0   6   0]
 [172   5   2 762  53   0   4   0   2   0]
```

```
[ 47   0  17  20 904   0   8   0   4   0]
[  2   0   8  33  12  19   2 857  10  57]
[325   1  14  19 623   0   7   0  11   0]
[  1   0   0   1   0   0   0 915   0  83]
[ 12   0  55   6  35   1   3   5 882   1]
[  3   0   0   6   3   0   0  69   1 918]]
```

```
disp = ConfusionMatrixDisplay(confusion_matrix=CM, display_labels=np.arange(10))

disp.plot( values_format ='.3g')
plt.show()
```



✓ 0 сек. выполнено в 03:24