```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Conv2D, Input
from tensorflow.keras.layers import GlobalMaxPooling2D, MaxPooling2D, AveragePooling2D, Gl

from tensorflow.keras.layers import Dense, Flatten, Concatenate

from tensorflow.keras.utils import plot_model, to_categorical
from tensorflow.keras.datasets import cifar10

import matplotlib.pyplot as plt
import os
```

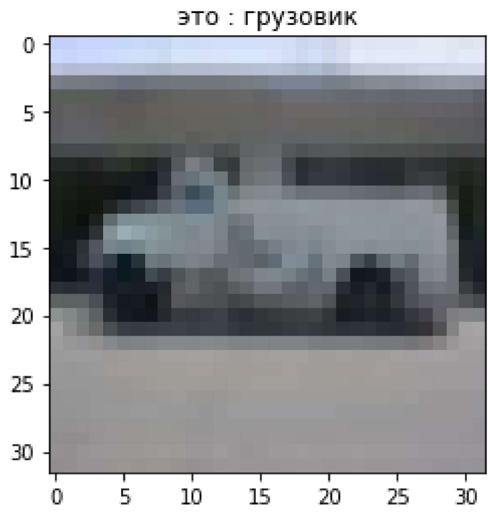
```
# установка параметров нейросети
batch_size = 32
num_classes = 10
epochs = 5
data_augmentation = False
num_predictions = 20
```

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'тренировочные примеры')
print(x_test.shape[0], 'тестовые примеры')

# преобразование матрицы чисел 0-9 в бинарную матрицу чисел 0-1
#трансформация лейблов в one-hot encoding
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 4s 0us/step
170508288/170498071 [==============================] - 4s 0us/step
x_train shape: (50000, 32, 32, 3)
50000 тренировочные примеры
10000 тестовые примеры
```

```
classes=['самолет', 'автомобиль', 'птица', 'кот', 'олень', 'собака', 'лягушка', 'лошадь',
```

```
N = 110

plt.imshow(x_train[N][:,:,:])
plt.title('это : '+classes[np.argmax(y_train[N,:])])
plt.show()
```




```
# изменение размерности массива в 4D массив
x_train = x_train.reshape(x_train.shape[0], 32,32,3)
x_test = x_test.reshape(x_test.shape[0], 32,32,3)
```

AlexNet

```
from tensorflow.keras.models import Model
# инициализация  модели
input1= keras.layers.Input(shape=(32,32,3))
# первый сверточный слой
x1 = keras.layers.Conv2D(120, kernel_size=(5, 5), strides=(1, 1), activation='tanh',  padd
print(f'размер x1 : {x1.shape}')
# второй пуллинговый слой
x2 = keras.layers.MaxPooling2D(pool_size=(2, 2), strides=(1, 1), padding='same')(x1)
print(f'размер x2 : {x2.shape}')
# третий сверточный слой
x3 = keras.layers.Conv2D(120, kernel_size=(2, 2), strides=(1, 1), activation='tanh', paddi
print(f'размер x3: {x3.shape}')
# четвертый пуллинговый слой
x4 = keras.layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same')(x3)
print(f'размер x4: {x4.shape}')
# пятый  слой
x5 = keras.layers.Conv2D(120, kernel_size=(5, 5), strides=(1, 1), activation='tanh', paddi
print(f'размер x5: {x5.shape}')
x6 = keras.layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid')(x5)
print(f'размер x6: {x6.shape}')
# пятый  слой
x7 = keras.layers.Conv2D(120, kernel_size=(5, 5), strides=(1, 1), activation='tanh', paddi
print(f'размер x7: {x7.shape}')
# сглаживание CNN выхода чтобы можно было его присоединить к полносвязногому слою
x8 = keras.layers.Flatten()(x7)
print(f'размер x8: {x8.shape}')
# шестой полносвязный слой
x9 = keras.layers.Dense(256, activation='tanh')(x8)
```

```
# выходной слой с функцией активации softmax
out_x = keras.layers.Dense(10, activation='softmax')(x9)



# Соберем полную модель сети от входа к выходу
model1 = Model(inputs = input1, outputs = out_x)
# сделаем несколько промежуточных выходов (через них посмотрим , что происходит в сети)
model3 = Model(inputs = input1, outputs = x3)
model5 = Model(inputs = input1, outputs = x5)
# компилияция модели
model1.compile(loss=keras.losses.categorical_crossentropy, optimizer= 'Adam', metrics=["ac

# Обучаем модель
hist = model1.fit(x=x_train,y=y_train, epochs=100, batch_size=128, validation_data=(x_test

test_score = model1.evaluate(x_test, y_test)
print("Test loss {:.2f}, accuracy {:.2f}%".format(test_score[0], test_score[1] * 100))
```

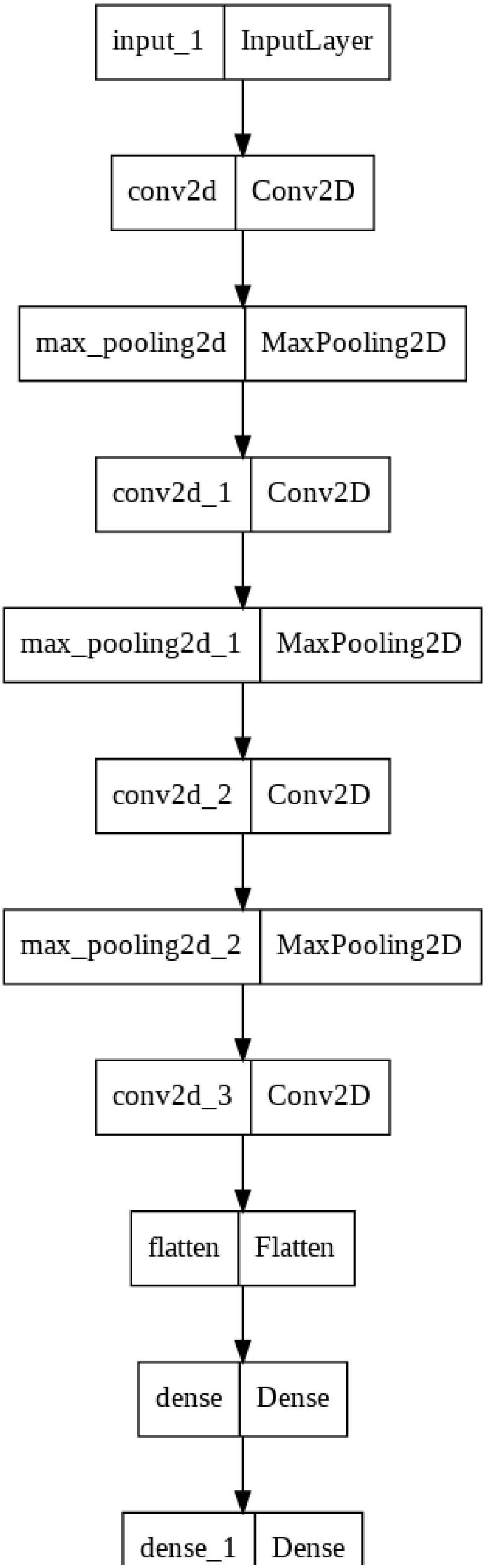
```
391/391 [==============================] - 37s 95ms/step - loss: 0.2803 - accurac
Epoch 74/100
391/391 [==============================] - 37s 96ms/step - loss: 0.3080 - accurac
Epoch 75/100
391/391 [==============================] - 38s 96ms/step - loss: 0.2822 - accurac
Epoch 76/100
391/391 [==============================] - 38s 96ms/step - loss: 0.2642 - accurac
Epoch 77/100
391/391 [==============================] - 38s 97ms/step - loss: 0.3179 - accurac
Epoch 78/100
391/391 [==============================] - 38s 96ms/step - loss: 0.3156 - accurac
Epoch 79/100
391/391 [==============================] - 37s 95ms/step - loss: 0.2959 - accurac
Epoch 80/100
391/391 [==============================] - 38s 96ms/step - loss: 0.2834 - accurac
Epoch 81/100
391/391 [==============================] - 38s 96ms/step - loss: 0.2857 - accurac
Epoch 82/100
391/391 [==============================] - 38s 97ms/step - loss: 0.2619 - accurac
Epoch 83/100
391/391 [==============================] - 38s 96ms/step - loss: 0.2863 - accurac
Epoch 84/100
391/391 [==============================] - 38s 97ms/step - loss: 0.3214 - accurac
Epoch 85/100
391/391 [==============================] - 37s 96ms/step - loss: 0.3087 - accurac
Epoch 86/100
391/391 [==============================] - 38s 96ms/step - loss: 0.2986 - accurac
Epoch 87/100
391/391 [==============================] - 38s 97ms/step - loss: 0.2925 - accurac
Epoch 88/100
391/391 [==============================] - 38s 97ms/step - loss: 0.3111 - accurac
Epoch 89/100
391/391 [==============================] - 38s 97ms/step - loss: 0.2831 - accurac
Epoch 90/100
391/391 [==============================] - 38s 97ms/step - loss: 0.2792 - accurac
Epoch 91/100
391/391 [==============================] - 38s 97ms/step - loss: 0.2862 - accurac
Epoch 92/100
391/391 [==============================] - 38s 97ms/step - loss: 0.2841 - accurac
Epoch 93/100
391/391 [==============================] - 38s 97ms/step - loss: 0.3164 - accurac
```

```
Epoch 94/100
391/391 [==============================] - 38s 97ms/step - loss: 0.3226 - accurac
Epoch 95/100
391/391 [==============================] - 38s 97ms/step - loss: 0.3422 - accurac
Epoch 96/100
391/391 [==============================] - 38s 96ms/step - loss: 0.3212 - accurac
Epoch 97/100
391/391 [==============================] - 38s 97ms/step - loss: 0.3081 - accurac
Epoch 98/100
391/391 [==============================] - 38s 96ms/step - loss: 0.2808 - accurac
Epoch 99/100
391/391 [==============================] - 38s 97ms/step - loss: 0.3102 - accurac
Epoch 100/100
391/391 [==============================] - 38s 96ms/step - loss: 0.2935 - accurac
313/313 [==============================] - 4s 13ms/step - loss: 1.1896 - accuracy
Test loss 1.19, accuracy 69.62%
```

```
model1.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 32, 32, 3)]       0

 conv2d (Conv2D)             (None, 32, 32, 120)       9120

 max_pooling2d (MaxPooling2D  (None, 32, 32, 120)      0
 )

 conv2d_1 (Conv2D)           (None, 32, 32, 120)       57720

 max_pooling2d_1 (MaxPooling  (None, 16, 16, 120)      0
 2D)

 conv2d_2 (Conv2D)           (None, 16, 16, 120)       360120

 max_pooling2d_2 (MaxPooling  (None, 8, 8, 120)        0
 2D)

 conv2d_3 (Conv2D)           (None, 4, 4, 120)         360120

 flatten (Flatten)           (None, 1920)              0

 dense (Dense)               (None, 256)               491776

 dense_1 (Dense)             (None, 10)                2570

=================================================================
Total params: 1,281,426
Trainable params: 1,281,426
Non-trainable params: 0
_________________________________________________________________
```

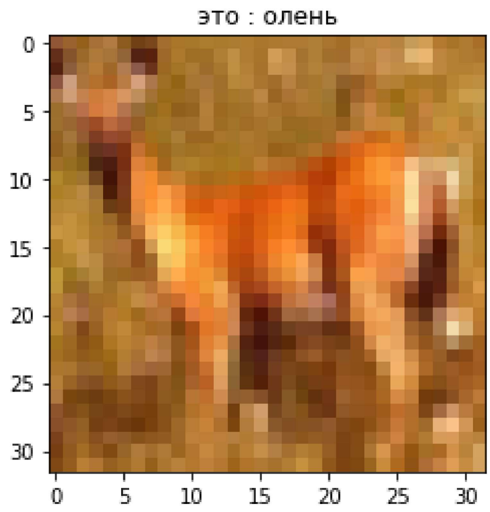
```
plot_model(model1,to_file='new_model-all.png')
```


input_1 | InputLayer
conv2d | Conv2D
max_pooling2d | MaxPooling2D
conv2d_1 | Conv2D
max_pooling2d_1 | MaxPooling2D
conv2d_2 | Conv2D
max_pooling2d_2 | MaxPooling2D
conv2d_3 | Conv2D
flatten | Flatten
dense | Dense
dense_1 | Dense

```
y_pred = model1.predict(x_test)

# y_pred[10].max()
n = 110
print(classes[list(y_pred[n]).index(max(y_pred[n]))])
```

```
собака
```

```
N = 110

plt.imshow(x_test[N][:,:,:])
plt.title('это : '+classes[np.argmax(y_test[N,:])])
plt.show()
```



Вывод по AlexNet

после изменения п пулинговом слое с average на maxpooling и добавления еще двух слоев , кол-во итераций = 25, оптимайзер был изменен на Adam т.к. на прошлом уроке он показал лучший результат, точность на train выросла до 97% а на test до 65%
уличение эпох до 100 и увелечение нейронов сильного прироста точности на тесте не дало, 69%, очень долго обучалась сеть, возможно если обучать еще дольше можно достигнуть нужного результата

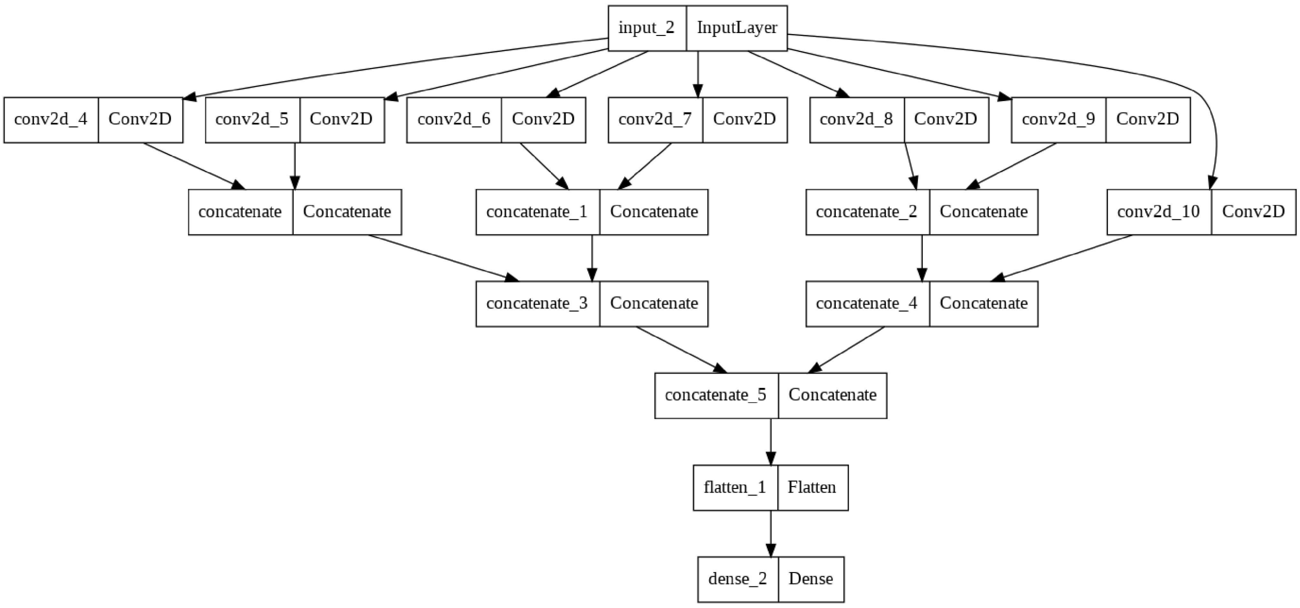
```
from tensorflow.python.ops.gen_dataset_ops import multi_device_iterator_get_next_from_shar
first_input = Input(shape=(32,32,3 ))
x11= Conv2D(128,3,activation='relu',padding = 'same')(first_input)
#x11= Flatten()(x11)
first_dense = x11# Dense(10, )(x11)

#second_input = Input(shape=(28,28,1 ))
x22= Conv2D(128,5,activation='relu',padding = 'same')(first_input)
#x22= Flatten()(x22)
second_dense = x22 #Dense(10, )(x22)

x111= Conv2D(128,3,activation='tanh',padding = 'same')(first_input)
```

```
    #x11= Flatten()(x11)
    first_dense2 = x111# Dense(10, )(x11)

    #second_input = Input(shape=(28,28,1 ))
    x222= Conv2D(128,5,activation='tanh',padding = 'same')(first_input)
    #x22= Flatten()(x22)
    second_dense2 = x222 #Dense(10, )(x22)

    x1111= Conv2D(128,3,activation='tanh',padding = 'same')(first_input)
    #x11= Flatten()(x11)
    first_dense3 = x1111# Dense(10, )(x11)

    #second_input = Input(shape=(28,28,1 ))
    x2222= Conv2D(128,5,activation='tanh',padding = 'same')(first_input)
    #x22= Flatten()(x22)
    second_dense3 = x2222 #Dense(10, )(x22)

    merge_one = Concatenate(   )([first_dense, second_dense])
    merge_two = Concatenate(   )([first_dense2, second_dense2])
    merge_three = Concatenate(   )([first_dense3, second_dense3])

    third_input = Input(shape=(32,32,3 ))
    x33= Conv2D(10,1,activation='relu',padding = 'same')(first_input)
    #x33= Flatten()(x33)
    #x33 = Dense(10, )(x33)
    merge_four = Concatenate( axis=-1)([merge_one, merge_two])
    merge_five = Concatenate( axis=-1)([ merge_three, x33])
    merge_six = Concatenate( axis=-1)([ merge_four, merge_five])
    x8 = keras.layers.Flatten()(merge_six)
    print(f'размер x8: {x8.shape}')

    merge_seven=Dense(10, activation='softmax')(x8)


    model_stek = Model(inputs=first_input, outputs=merge_seven)
    #model_stek = Model(inputs=[first_input, second_input, third_input], outputs=merge_two)
    ada_grad = tf.keras.optimizers.Adagrad(lr=0.1, epsilon=1e-08, decay=0.0)
    # model_stek.compile(optimizer=ada_grad, loss=tf.keras.losses.CategoricalCrossentropy(),
                  # metrics=['accuracy'])
    model_stek.compile(loss=keras.losses.categorical_crossentropy, optimizer= 'Adam', metrics=


    hist = model_stek.fit(x=x_train,y=y_train, epochs=100, batch_size=128, validation_data=(x_

    test_score = model1.evaluate(x_test, y_test)
    print("Test loss {:.2f}, accuracy {:.2f}%".format(test_score[0], test_score[1] * 100))
```

```
391/391 [==============================] - 70s 179ms/step - loss: 0.0499 - accura
Epoch 74/100
391/391 [==============================] - 70s 179ms/step - loss: 0.0400 - accura
Epoch 75/100
391/391 [==============================] - 72s 184ms/step - loss: 0.1169 - accura
Epoch 76/100
391/391 [==============================] - 70s 179ms/step - loss: 0.0139 - accura
Epoch 77/100
```

```
391/391 [==============================] - 72s 183ms/step - loss: 0.0041 - accura
Epoch 78/100
391/391 [==============================] - 70s 179ms/step - loss: 0.0049 - accura
Epoch 79/100
391/391 [==============================] - 70s 179ms/step - loss: 0.0163 - accura
Epoch 80/100
391/391 [==============================] - 70s 179ms/step - loss: 0.2115 - accura
Epoch 81/100
391/391 [==============================] - 70s 179ms/step - loss: 0.0092 - accura
Epoch 82/100
391/391 [==============================] - 70s 179ms/step - loss: 0.0036 - accura
Epoch 83/100
391/391 [==============================] - 72s 185ms/step - loss: 0.0020 - accura
Epoch 84/100
391/391 [==============================] - 70s 180ms/step - loss: 0.0016 - accura
Epoch 85/100
391/391 [==============================] - 70s 180ms/step - loss: 0.0013 - accura
Epoch 86/100
391/391 [==============================] - 70s 180ms/step - loss: 0.0059 - accura
Epoch 87/100
391/391 [==============================] - 71s 181ms/step - loss: 0.1339 - accura
Epoch 88/100
391/391 [==============================] - 70s 180ms/step - loss: 0.0404 - accura
Epoch 89/100
391/391 [==============================] - 70s 180ms/step - loss: 0.0232 - accura
Epoch 90/100
391/391 [==============================] - 70s 179ms/step - loss: 0.0137 - accura
Epoch 91/100
391/391 [==============================] - 70s 179ms/step - loss: 0.0087 - accura
Epoch 92/100
391/391 [==============================] - 70s 179ms/step - loss: 0.0088 - accura
Epoch 93/100
391/391 [==============================] - 70s 179ms/step - loss: 11.6705 - accur
Epoch 94/100
391/391 [==============================] - 70s 180ms/step - loss: 0.0417 - accura

Epoch 95/100
391/391 [==============================] - 72s 184ms/step - loss: 0.0229 - accura
Epoch 96/100
391/391 [==============================] - 70s 179ms/step - loss: 0.0084 - accura
Epoch 97/100
391/391 [==============================] - 70s 179ms/step - loss: 0.0164 - accura
Epoch 98/100
391/391 [==============================] - 72s 183ms/step - loss: 0.0075 - accura
Epoch 99/100
391/391 [==============================] - 70s 180ms/step - loss: 0.0023 - accura
Epoch 100/100
391/391 [==============================] - 70s 179ms/step - loss: 0.0191 - accura
313/313 [==============================] - 4s 13ms/step - loss: 1.1896 - accuracy
Test loss 1.19, accuracy 69.62%
```

```
plot_model(model_stek,'model_stek.png')
```



Вывод по Сетям со сложными конструкциями

после добавления большего кол-ва слоев и увеличения нейронов до 50 сеть стала дольше обучаться, результат на тесте доходит до 60% а потом с очень маленьким шагом то выше, то ниже, но потихоньку через одну или две итерации всё таки вырастает
После увелечения эпох до 100 и большего кол-ва нейронов результат вырос до 69%