Implantação do README Github LCAD Deep Learning 2017/2

Autores: Renan Mantuanelli de Aquino e Deivison Augusto da Vitória

Passo a passo de como criar uma CNN (Convolutional Neural Network) para identificação de carro, gato e cachorro usando Keras e Tensor Flow baseado na AlexNet e ZFNet usando Windows 10.

1. Instalar anaconda phyton
   - https://www.continuum.io/downloads)
2. Instalar spyder
   - Abrir anaconda navegator e instalar spyder
3. Instalar theano
   - Abrir anaconda prompt e instalar theano: pip install theano
4. Instalar tensorflow
   - Abrir anaconda prompt e instalar theano: pip install tensorflow
5. Instalar keras
   - Abrir anaconda prompt e instalar keras: pip install keras
6. Atualizar todos os programas recém instalados e outros pacotes do Anaconda
   - Abrir anaconda prompt e atualizar: conda update –all
7. Instalar o Vstudio 2015
8. Instalar cuda_8.0.61_win10
9. Instalar cuda_8.0.61.2_windows
10. Incluir caminho no Path:
    - edit system variables
    - Environment variables
    - System variables
    - Path Edit
    - New (inserir na última linha caminho que está instalado o cuda ex.: C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\bin)
11. Instalar GPU para utilização no Tensorflow
    - https://www.tensorflow.org/install/install_windows
    - Criar ambiente conda com nome tensorflow (abrir anaconda prompt)
      - ✓ conda create -n tensorflow python=3.5
    - Ativar ambiente conda (abrir anaconda prompt)
      - ✓ activate tensorflow
    - Atualizar tensorflow (abrir anaconda prompt)
      - ✓ pip install --ignore-installed --upgrade tensorflow
    - Instalar GPU versão do tensorflow
      - ✓ pip install --ignore-installed --upgrade tensorflow-gpu
    - Testar sucesso instalação digitando os comandos abaixo no spyder
      - ✓ Abrir spyder (phynton)
      - ✓ import tensorflow as tf
        hello = tf.constant('Hello, TensorFlow!')
        sess = tf.Session()
        print(sess.run(hello))
    - Se a instalação foi concluída com sucesso a saída (Console Phyton - Spyder) irá retornar *Hello, TensorFlow!*

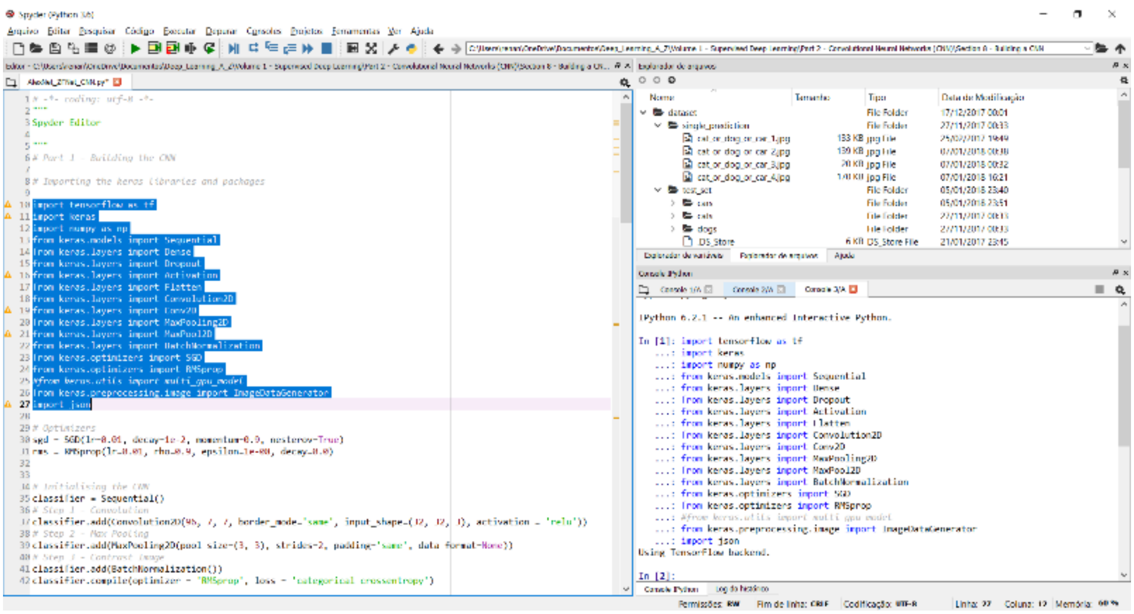
12. Criar a rede CNN baseado na ZFnet

```
"""
Spyder Editor
"""
```

 # Part 1 – Construindo a rede Aquino&VitóriaNet (CNN) baseado na ZFNET

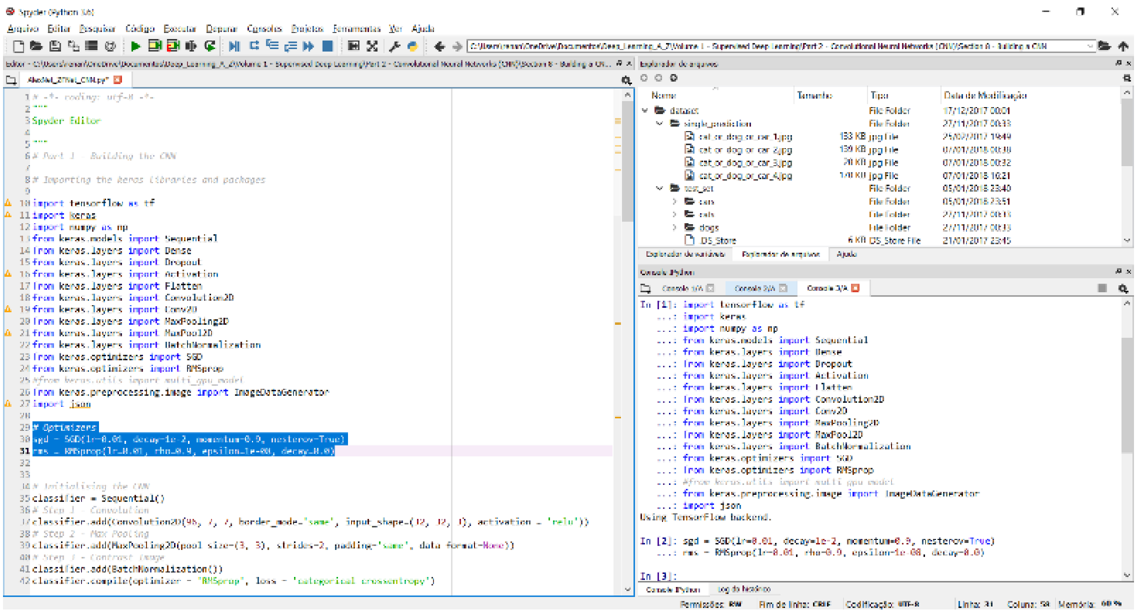
Importando o tensorflow, keras, pacotes e bibliotecas

```
import tensorflow as tf
import keras
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Activation
from keras.layers import Flatten
from keras.layers import Convolution2D
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import MaxPool2D
from keras.layers import BatchNormalization
from keras.optimizers import SGD
from keras.optimizers import RMSprop
from keras.utils import multi_gpu_model
from keras.preprocessing.image import ImageDataGenerator
import json
```



Definindo as variáveis e funções de otimizações para melhorar aprendizado no treinamento

```
sgd = SGD(lr=0.01, decay=1e-2, momentum=0.9, nesterov=True)

rms = RMSprop(lr=0.01, rho=0.9, epsilon=1e-08, decay=0.0)
```



Inicializando a rede Aquino&VitóriaFNET CNN

O tipo da rede será sequencial que é uma característica da CNN

classifier = Sequential()

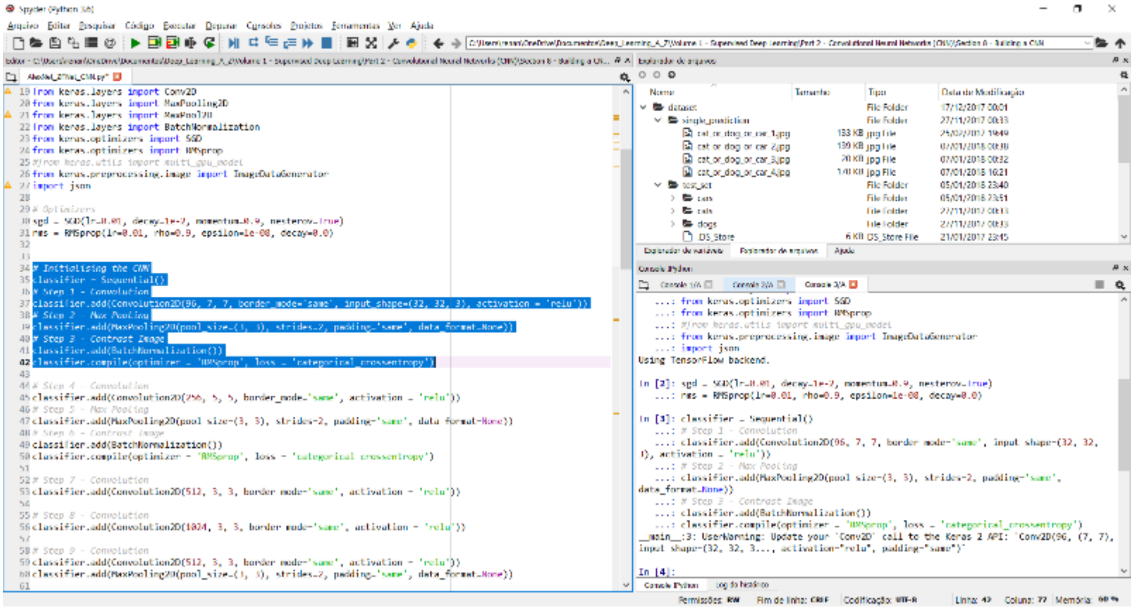
Step 1 – Convolution – 96 camadas, kernel 7x7, imagem entrada 32x32x3(RGB), função de ativação ReLU

classifier.add(Convolution2D(96, 7, 7, border_mode='same', input_shape=(32, 32, 3), activation = 'relu'))

# Step 2 - Max Pooling – tamanho 3x3, passo2
classifier.add(MaxPooling2D(pool_size=(3, 3), strides=2, padding='same', data_format=None))

Step 3 - Contrast Image – aplicando operação de contraste local de normalização na saída do mapa de caracteríscica (feature map)

classifier.add(BatchNormalization())
classifier.compile(optimizer = 'RMSprop', loss = 'categorical_crossentropy')



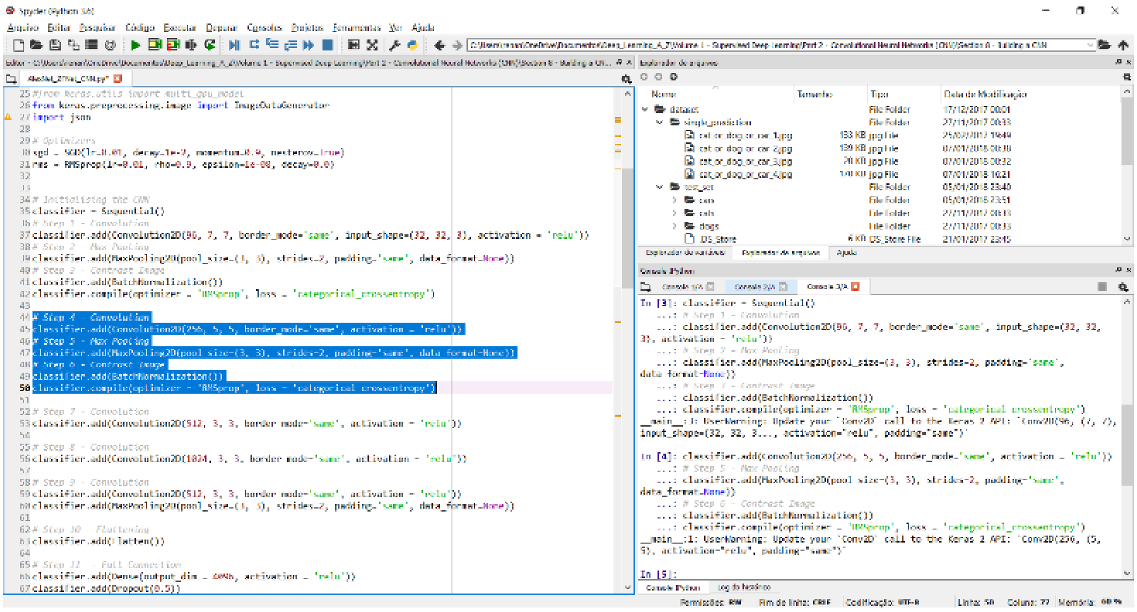
Step 4 – Convolution - 256 camadas, kernel 5x5, imagem entrada 16x16x256 função de ativação ReLU

classifier.add(Convolution2D(256, 5, 5, border_mode='same', activation = 'relu'))

# Step 5 - Max Pooling - tamanho 3x3, passo2
classifier.add(MaxPooling2D(pool_size=(3, 3), strides=2, padding='same', data_format=None))

Step 6 - Contrast Image - aplicando operação de contraste local de normalização na saída do mapa de caracteríscica (feature map)

classifier.add(BatchNormalization())
classifier.compile(optimizer = 'RMSprop', loss = 'categorical_crossentropy')



Step 7 – Convolution - 512 camadas, kernel 3x3, imagem entrada 8x8x512, função de ativação ReLU

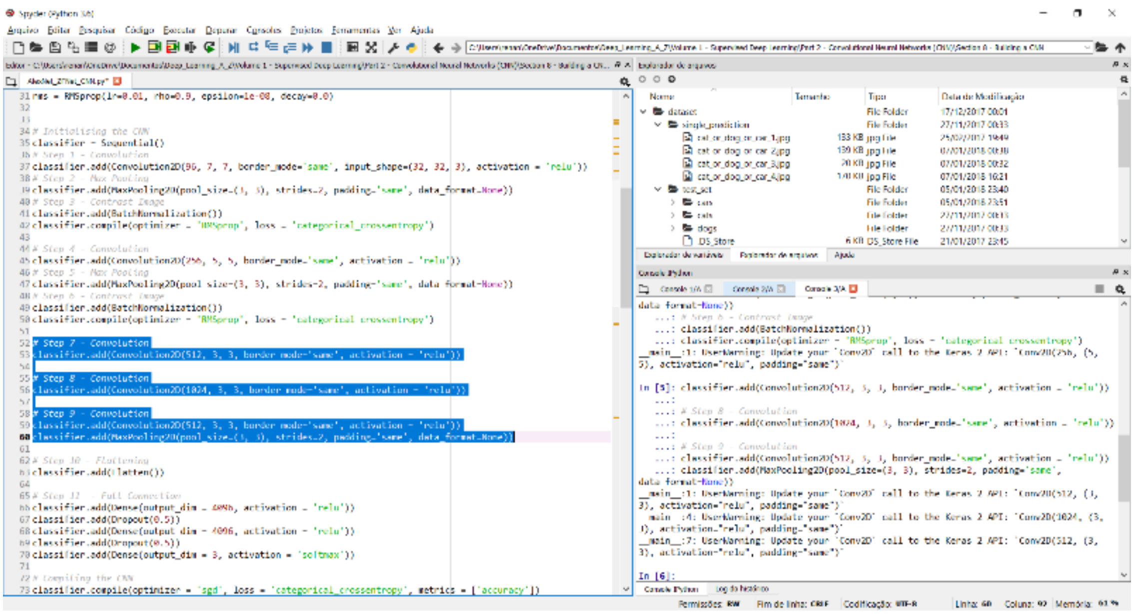
classifier.add(Convolution2D(512, 3, 3, border_mode='same', activation = 'relu'))

Step 8 - Convolution - 1024 camadas, kernel 3x3, imagem entrada 8x8x1024, função de ativação ReLU

classifier.add(Convolution2D(1024, 3, 3, border_mode='same', activation = 'relu'))

Step 9 - Convolution - 512 camadas, kernel 3x3, imagem entrada 8x8x512, função de ativação ReLU

classifier.add(Convolution2D(512, 3, 3, border_mode='same', activation = 'relu'))



# Step 10 - Max Pooling - tamanho 3x3, passo2
classifier.add(MaxPooling2D(pool_size=(3, 3), strides=2, padding='same', data_format=None))

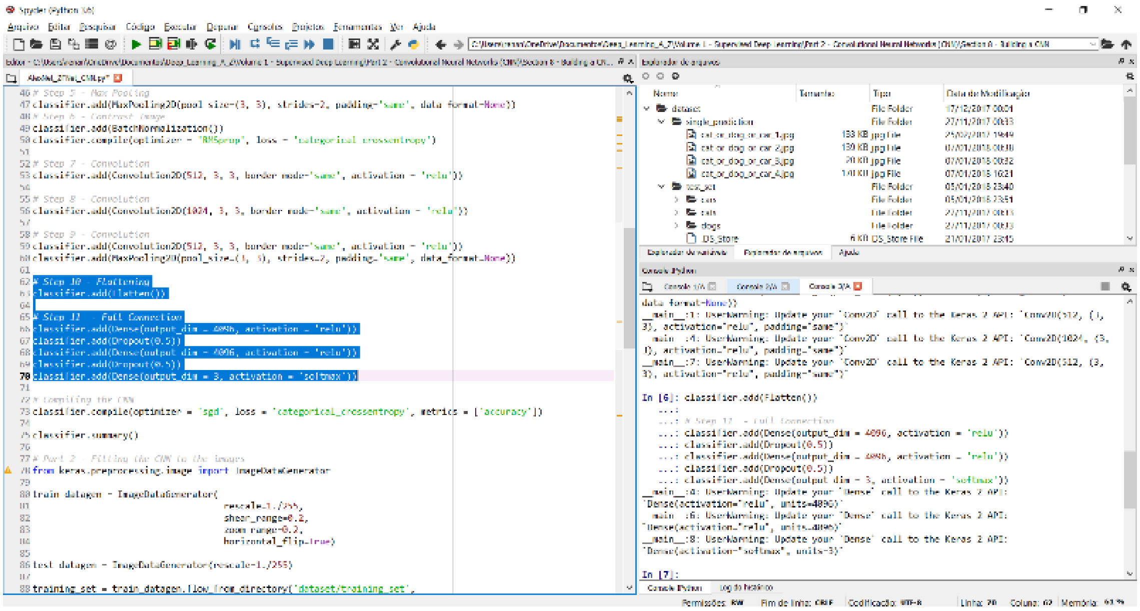
Step 11 – Flattening – preparação da saída da última etapa de convolução conforme necessário para entrada na camada primeira totalmente conectada com 8192 saídas (dimensões).

classifier.add(Flatten())

Step 12 - Full Connection – primeira camada totalmente conectada com 4096 dimensões, 33.558.528 parâmetros e função de ativação ReLU, desligamento de 50% dos neurônios (dropout). A segunda camada totalmente conectada com 4096 dimensões, 16.781.312 parâmetros e função de ativação ReLU, desligamento de 50% dos neurônios (dropout). A última camada classificadora do tipo softmax com três saídas (carro, gato ou cachorro) com 12.291 parâmetros.

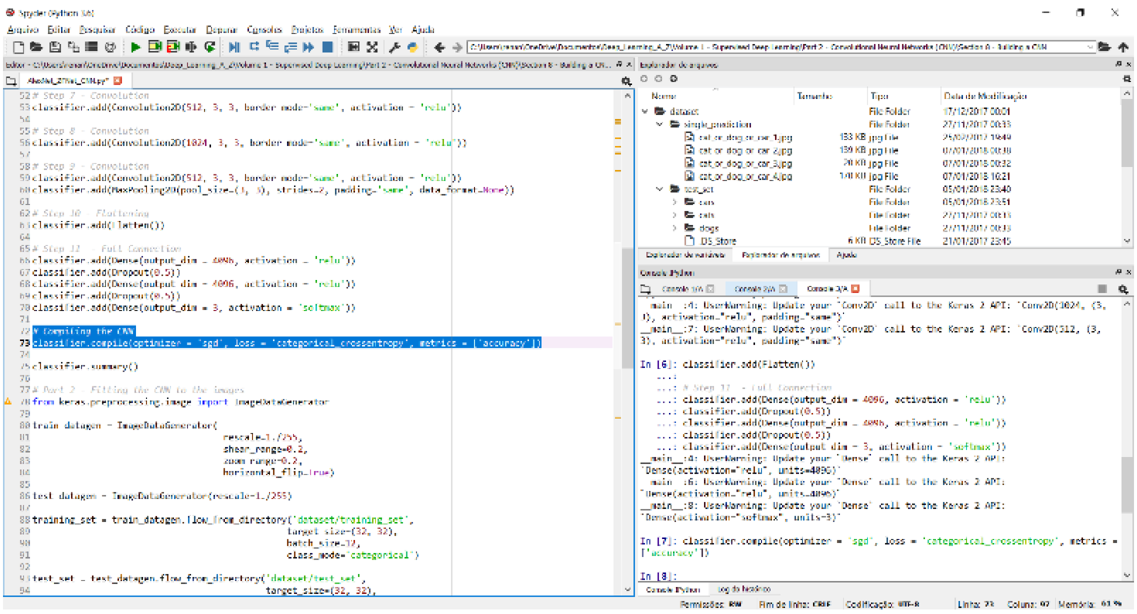
No caso da ZFNet a saída da última camada seria 1000, em funções da possibilidade de classificação de 1000 classes diferentes.

```
classifier.add(Dense(output_dim = 4096, activation = 'relu'))
classifier.add(Dropout(0.5))
classifier.add(Dense(output_dim = 4096, activation = 'relu'))
classifier.add(Dropout(0.5))
classifier.add(Dense(output_dim = 3, activation = 'softmax'))
```

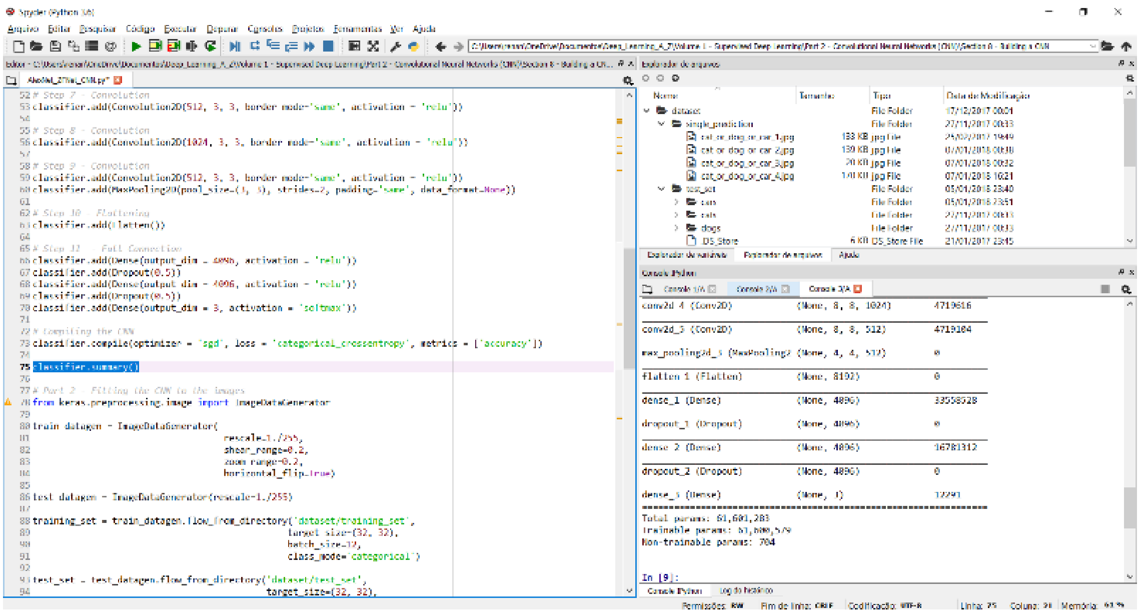


Compiling the CNN – Compilando a rede neural para classificação final (acuracidade) e verificação da arquitetura da rede

```
classifier.compile(optimizer = 'sgd', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```



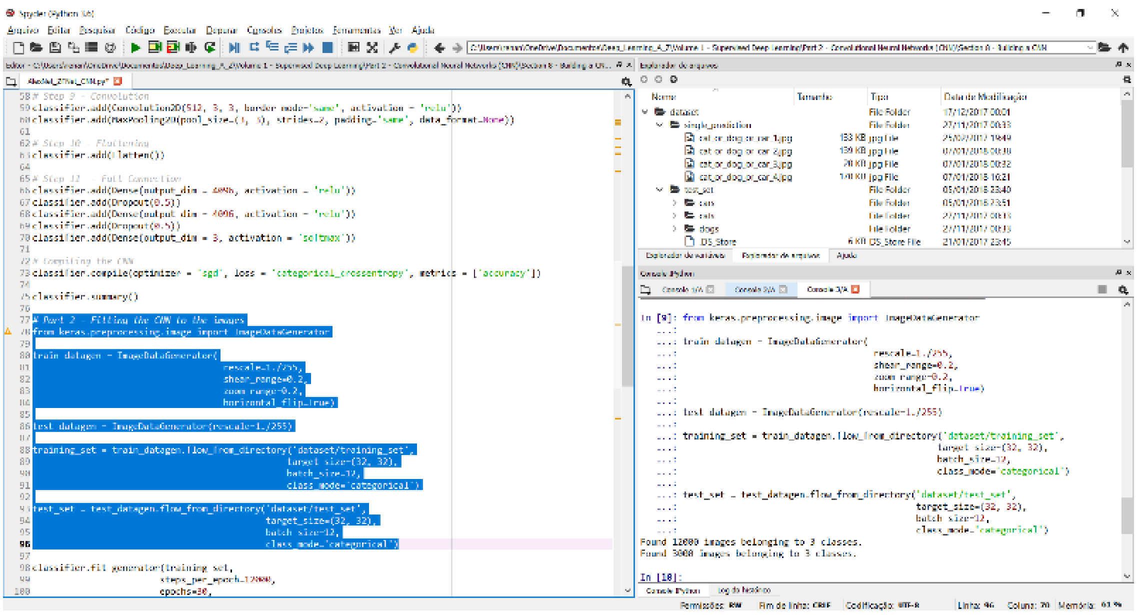
```
classifier.summary()
```



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 96)        14208
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 96)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 16, 16, 96)        384
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 256)       614656
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 256)         0
_________________________________________________________________
batch_normalization_2 (Batch (None, 8, 8, 256)         1024
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 8, 8, 512)         1180160
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 1024)        4719616
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 512)         4719104
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 512)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              33558528
_________________________________________________________________
dropout_1 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_2 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dropout_2 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_3 (Dense)              (None, 3)                 12291
=================================================================
Total params: 61,601,283
Trainable params: 61,600,579
Non-trainable params: 704
_________________________________________________________________
```

 # Part 2 – Fazendo a ligação da rede neural com as imagens para treino, teste/validação

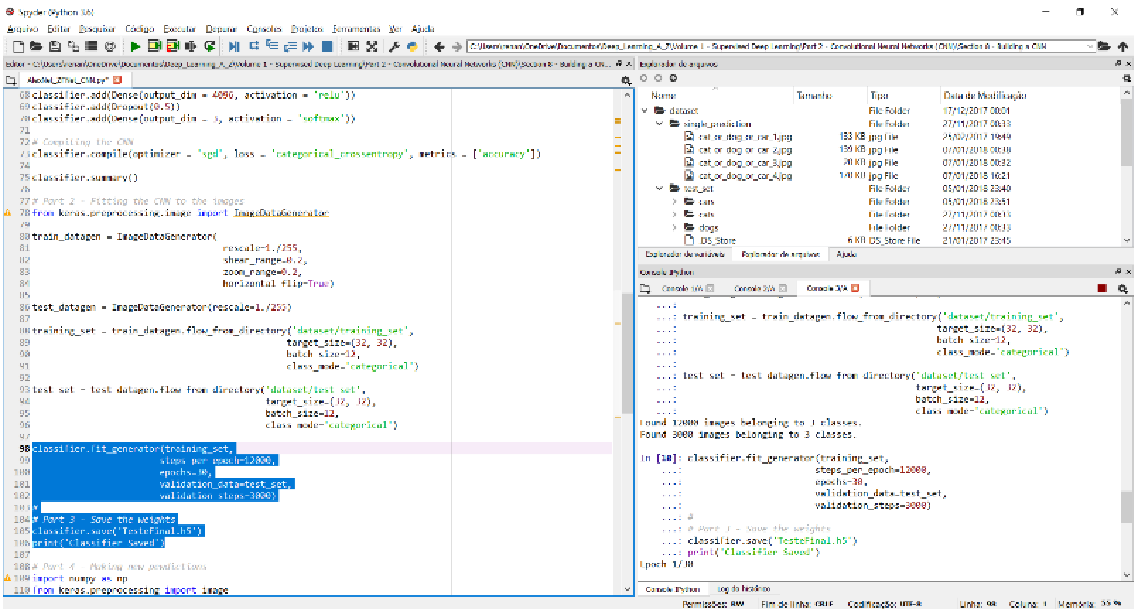
```
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(
                        rescale=1./255,
                        shear_range=0.2,
                        zoom_range=0.2,
                        horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1./255)

training_set = train_datagen.flow_from_directory('dataset/training_set',
                                target_size=(32, 32),
                                batch_size=12,
                                class_mode='categorical')

test_set = test_datagen.flow_from_directory('dataset/test_set',
                                target_size=(32, 32),
                                batch_size=12,
                                class_mode='categorical')
```



```
classifier.fit_generator(training_set,
                steps_per_epoch=12000,
                epochs=30,
                validation_data=test_set,
                validation_steps=3000)
```

 # Part 3 – Salvando os pesos após treino (aprendizado "memória" da rede)

```
classifier.save('TesteFinal.h5')
print('Classifier Saved')
```

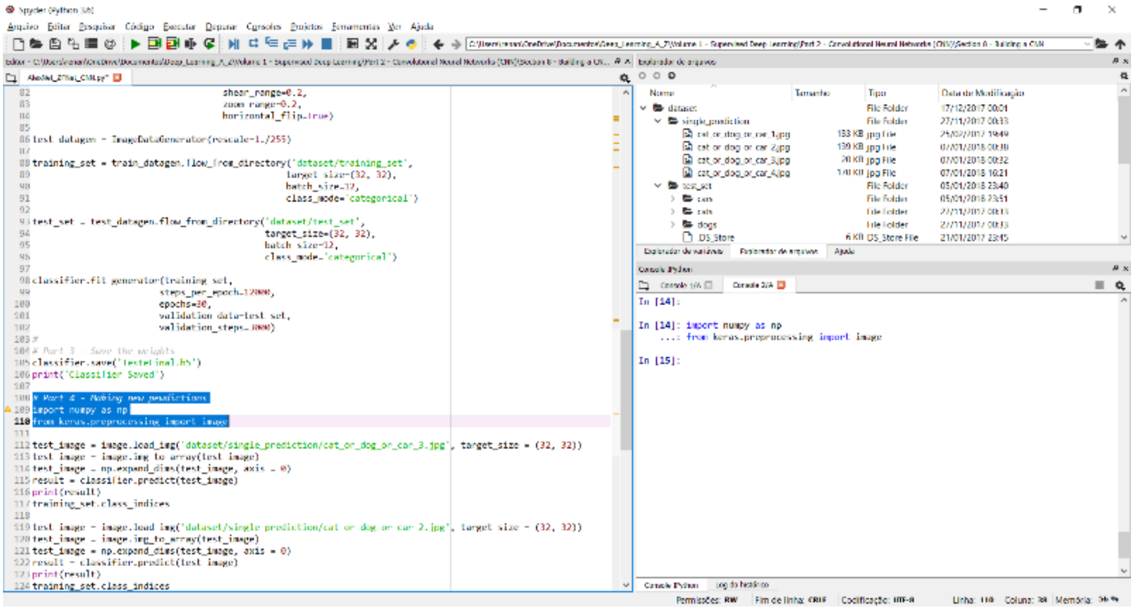


Epoch 1/30
12000/12000 [==============================] - 1599s 133ms/step - loss: 0.4411 - acc: 0.7928 - val_loss: 0.3240 - val_acc: 0.8569
Epoch 2/30
12000/12000 [==============================] - 1587s 132ms/step - loss: 0.2532 - acc: 0.8913 - val_loss: 0.2935 - val_acc: 0.8838
Epoch 3/30
12000/12000 [==============================] - 1588s 132ms/step - loss: 0.1570 - acc: 0.9369 - val_loss: 0.3353 - val_acc: 0.8817
Epoch 4/30
12000/12000 [==============================] - 1594s 133ms/step - loss: 0.0954 - acc: 0.9629 - val_loss: 0.3726 - val_acc: 0.8699
Epoch 5/30
12000/12000 [==============================] - 1549s 129ms/step - loss: 0.0615 - acc: 0.9771 - val_loss: 0.4296 - val_acc: 0.8810
Epoch 6/30
12000/12000 [==============================] - 1553s 129ms/step - loss: 0.0419 - acc: 0.9849 - val_loss: 0.5313 - val_acc: 0.8763
Epoch 7/30
12000/12000 [==============================] - 1553s 129ms/step - loss: 0.0315 - acc: 0.9888 - val_loss: 0.5310 - val_acc: 0.8803
Epoch 8/30
12000/12000 [==============================] - 1552s 129ms/step - loss: 0.0249 - acc: 0.9913 - val_loss: 0.4968 - val_acc: 0.8817
Epoch 9/30
12000/12000 [==============================] - 1548s 129ms/step - loss: 0.0188 - acc: 0.9936 - val_loss: 0.4857 - val_acc: 0.8947
Epoch 10/30
12000/12000 [==============================] - 1559s 130ms/step - loss: 0.0155 - acc: 0.9945 - val_loss: 0.4752 - val_acc: 0.8896
Epoch 11/30
12000/12000 [==============================] - 1552s 129ms/step - loss: 0.0148 - acc: 0.9948 - val_loss: 0.5251 - val_acc: 0.8889
Epoch 12/30
12000/12000 [==============================] - 1548s 129ms/step - loss: 0.0121 - acc: 0.9956 - val_loss: 0.5172 - val_acc: 0.8886
Epoch 13/30

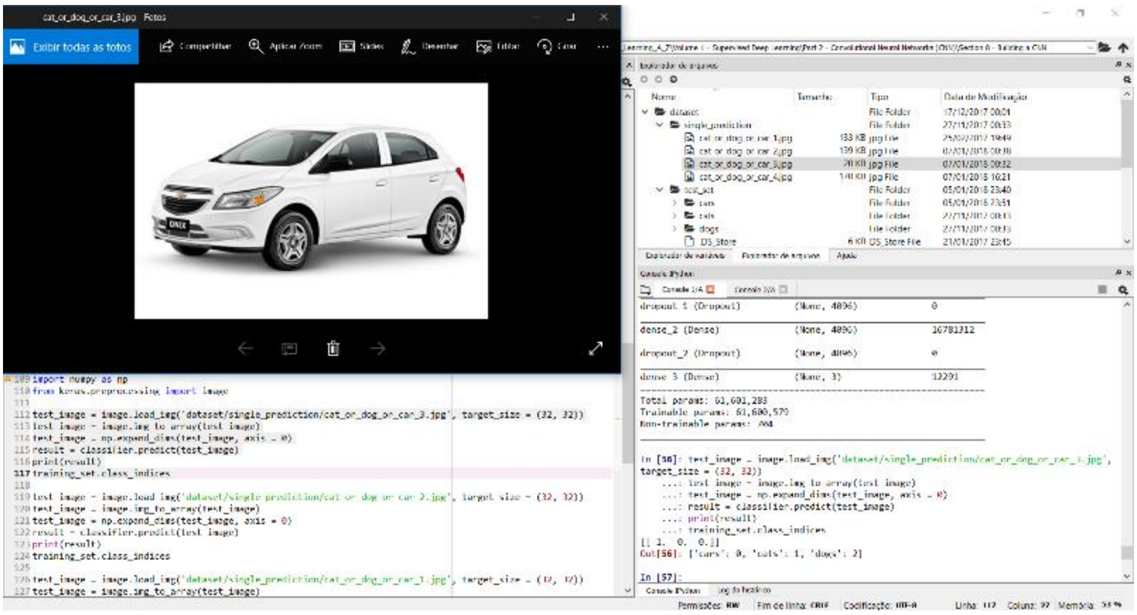
12000/12000 [==============================] - 1547s 129ms/step - loss: 0.0116 - acc: 0.9960 - val_loss: 0.5079 - val_acc: 0.8935
Epoch 14/30
12000/12000 [==============================] - 1547s 129ms/step - loss: 0.0081 - acc: 0.9971 - val_loss: 0.5743 - val_acc: 0.8947
Epoch 15/30
12000/12000 [==============================] - 1547s 129ms/step - loss: 0.0097 - acc: 0.9965 - val_loss: 0.5999 - val_acc: 0.8870
Epoch 16/30
12000/12000 [==============================] - 1547s 129ms/step - loss: 0.0085 - acc: 0.9972 - val_loss: 0.7481 - val_acc: 0.8732
Epoch 17/30
12000/12000 [==============================] - 1547s 129ms/step - loss: 0.0069 - acc: 0.9976 - val_loss: 0.5989 - val_acc: 0.8936
Epoch 18/30
12000/12000 [==============================] - 1548s 129ms/step - loss: 0.0070 - acc: 0.9977 - val_loss: 0.5413 - val_acc: 0.8943
Epoch 19/30
12000/12000 [==============================] - 1548s 129ms/step - loss: 0.0047 - acc: 0.9983 - val_loss: 0.6151 - val_acc: 0.8958
Epoch 20/30
12000/12000 [==============================] - 1548s 129ms/step - loss: 0.0067 - acc: 0.9977 - val_loss: 0.5877 - val_acc: 0.8902
Epoch 21/30
12000/12000 [==============================] - 1548s 129ms/step - loss: 0.0048 - acc: 0.9983 - val_loss: 0.5425 - val_acc: 0.8988
Epoch 22/30
12000/12000 [==============================] - 1546s 129ms/step - loss: 0.0053 - acc: 0.9983 - val_loss: 0.6114 - val_acc: 0.8904
Epoch 23/30
12000/12000 [==============================] - 1547s 129ms/step - loss: 0.0047 - acc: 0.9985 - val_loss: 0.5633 - val_acc: 0.8944
Epoch 24/30
12000/12000 [==============================] - 1547s 129ms/step - loss: 0.0042 - acc: 0.9986 - val_loss: 0.5868 - val_acc: 0.9000
Epoch 25/30
12000/12000 [==============================] - 1550s 129ms/step - loss: 0.0036 - acc: 0.9988 - val_loss: 0.6403 - val_acc: 0.8978
Epoch 26/30
12000/12000 [==============================] - 1553s 129ms/step - loss: 0.0037 - acc: 0.9988 - val_loss: 0.6719 - val_acc: 0.8914
Epoch 27/30
12000/12000 [==============================] - 1554s 130ms/step - loss: 0.0041 - acc: 0.9986 - val_loss: 0.5864 - val_acc: 0.8962
Epoch 28/30
12000/12000 [==============================] - 1555s 130ms/step - loss: 0.0038 - acc: 0.9988 - val_loss: 0.5740 - val_acc: 0.8957
Epoch 29/30
12000/12000 [==============================] - 1550s 129ms/step - loss: 0.0032 - acc: 0.9990 - val_loss: 0.6050 - val_acc: 0.8961
Epoch 30/30
12000/12000 [==============================] - 1546s 129ms/step - loss: 0.0037 - acc: 0.9987 - val_loss: 0.6404 - val_acc: 0.8955
Classifier Saved

 # Part 4 – Fazendo novas predições com a rede já treinada

import numpy as np
from keras.preprocessing import image

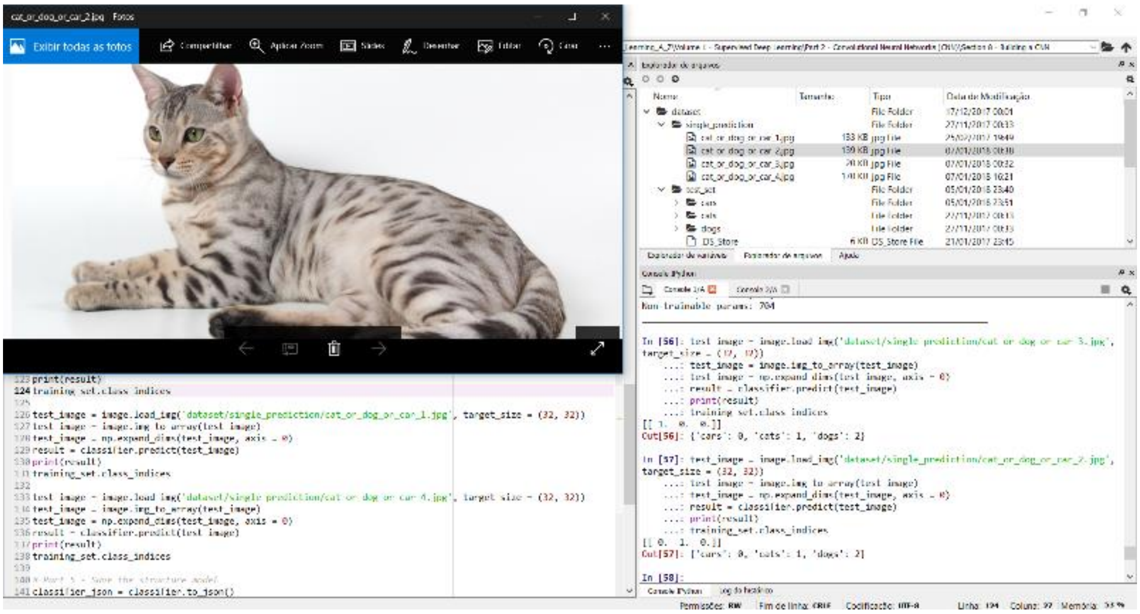


test_image = image.load_img('dataset/single_prediction/cat_or_dog_or_car_3.jpg', target_size = (32, 32))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = classifier.predict(test_image)
print(result)
training_set.class_indices

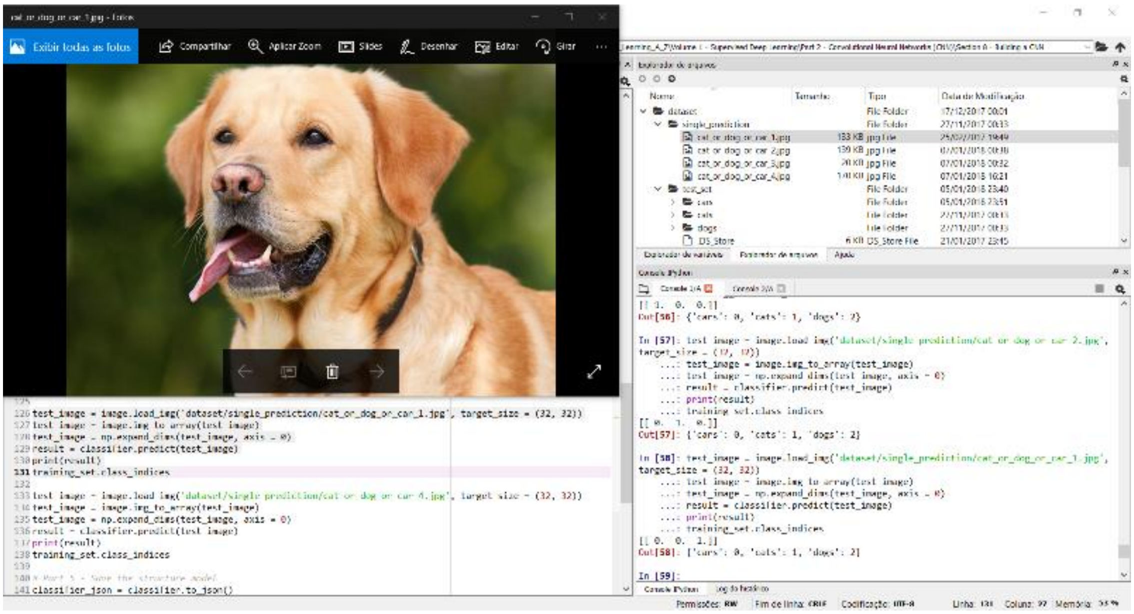


test_image = image.load_img('dataset/single_prediction/cat_or_dog_or_car_2.jpg', target_size = (32, 32))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = classifier.predict(test_image)

```
print(result)
training_set.class_indices
```

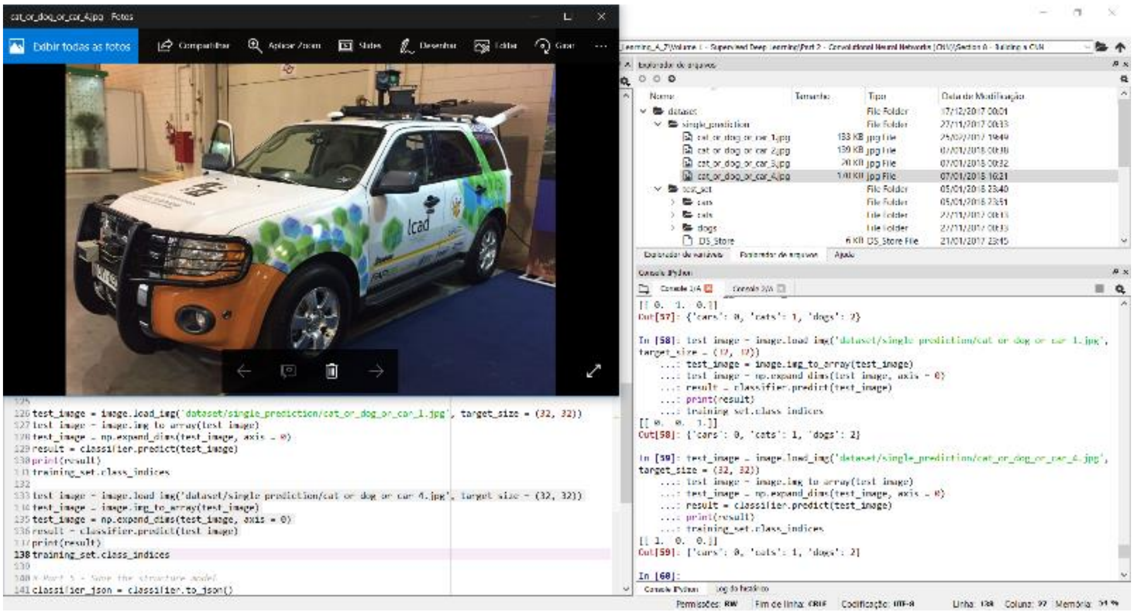


```
test_image = image.load_img('dataset/single_prediction/cat_or_dog_or_car_1.jpg',
target_size = (32, 32))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = classifier.predict(test_image)
print(result)
training_set.class_indices
```



```
test_image = image.load_img('dataset/single_prediction/cat_or_dog_or_car_4.jpg',
target_size = (32, 32))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = classifier.predict(test_image)
```

print(result)
training_set.class_indices



Part 5 – Salvando a estrutura do modelo

```
classifier_json = classifier.to_json()
with open("Aquino&VitótiaNetStructure.json", "w") as json_file:
    json_file.write(classifier_json)
```