Implementación de CNN, COVID19

Universidad Nacional de San Agustín de Arequipa
Ciencia de la Computación

Tópicos en Inteligencia Artificial
Cristia López

Brayan Maguiña del Castillo
Kevin Salazar Torres
Raúl Quispe Totocayo

May 24, 2020

Clasificación de pulmones con (Covid-Normales)

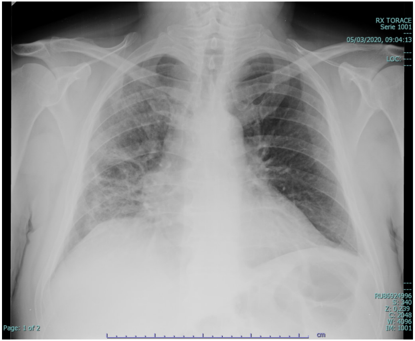


Figure: Con Covid

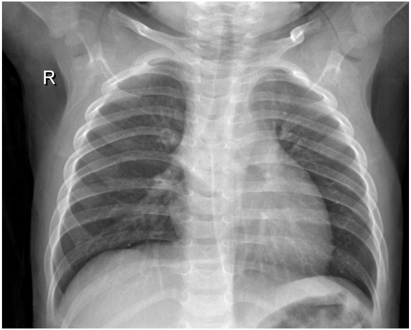


Figure: Normal

Librerías Usadas para construir el modelo

- Keras
- Tensorflow

Procesamiento de datos

Data set https://www.kaggle.com/nabeelsajid917/covid-19-x-ray-10000-images

- normal 28 imagenes
- covid 70 imagenes

InceptionV4

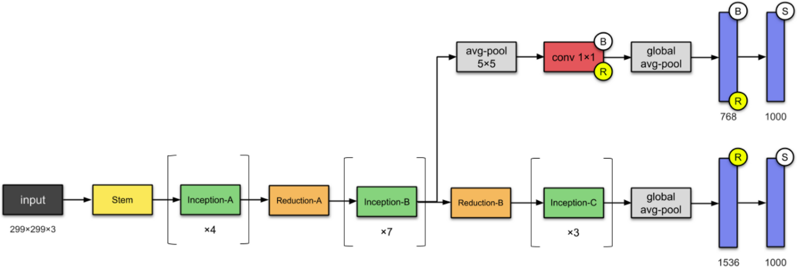


Figure: modelo general

Block Stem

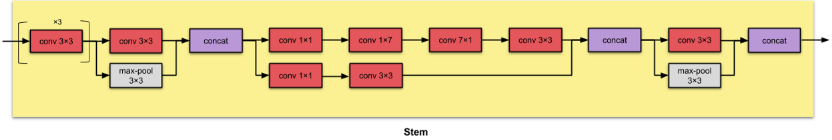
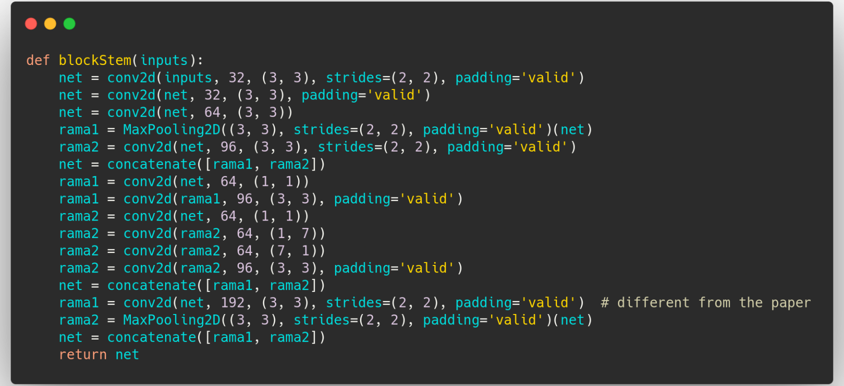


Figure: Stem

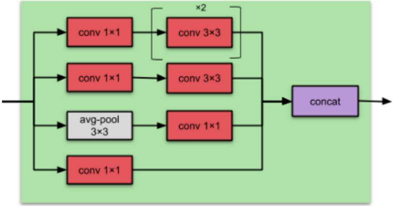
Block stem en keras



```
def blockStem(inputs):
    net = conv2d(inputs, 32, (3, 3), strides=(2, 2), padding='valid')
    net = conv2d(net, 32, (3, 3), padding='valid')
    net = conv2d(net, 64, (3, 3))
    rama1 = MaxPooling2D((3, 3), strides=(2, 2), padding='valid')(net)
    rama2 = conv2d(net, 96, (3, 3), strides=(2, 2), padding='valid')
    net = concatenate([rama1, rama2])
    rama1 = conv2d(net, 64, (1, 1))
    rama1 = conv2d(rama1, 96, (3, 3), padding='valid')
    rama2 = conv2d(net, 64, (1, 1))
    rama2 = conv2d(rama2, 64, (1, 7))
    rama2 = conv2d(rama2, 64, (7, 1))
    rama2 = conv2d(rama2, 96, (3, 3), padding='valid')
    net = concatenate([rama1, rama2])
    rama1 = conv2d(net, 192, (3, 3), strides=(2, 2), padding='valid')  # different from the paper
    rama2 = MaxPooling2D((3, 3), strides=(2, 2), padding='valid')(net)
    net = concatenate([rama1, rama2])
    return net
```

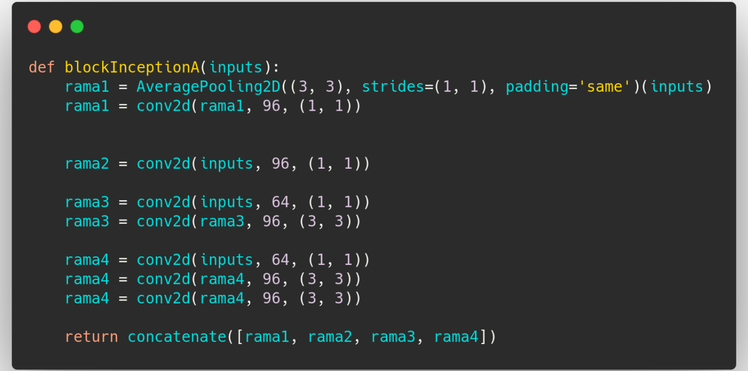
Figure: Implementación en keras

Block Inception A



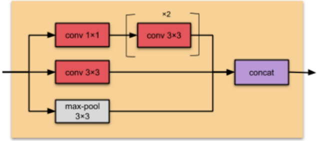
Inception-A

Block Inception A en keras



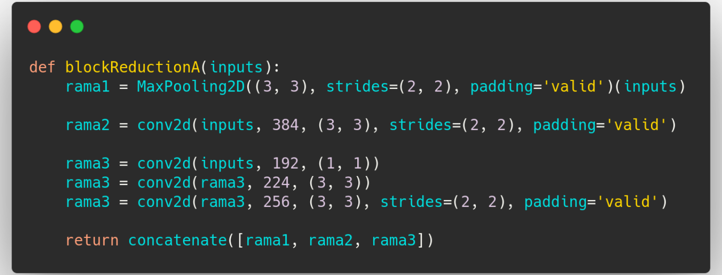
```python
def blockInceptionA(inputs):
    rama1 = AveragePooling2D((3, 3), strides=(1, 1), padding='same')(inputs)
    rama1 = conv2d(rama1, 96, (1, 1))


    rama2 = conv2d(inputs, 96, (1, 1))

    rama3 = conv2d(inputs, 64, (1, 1))
    rama3 = conv2d(rama3, 96, (3, 3))

    rama4 = conv2d(inputs, 64, (1, 1))
    rama4 = conv2d(rama4, 96, (3, 3))
    rama4 = conv2d(rama4, 96, (3, 3))

    return concatenate([rama1, rama2, rama3, rama4])
```

Block Reduction A



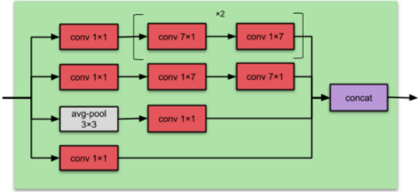
Reduction-A

Block Reduction A en keras



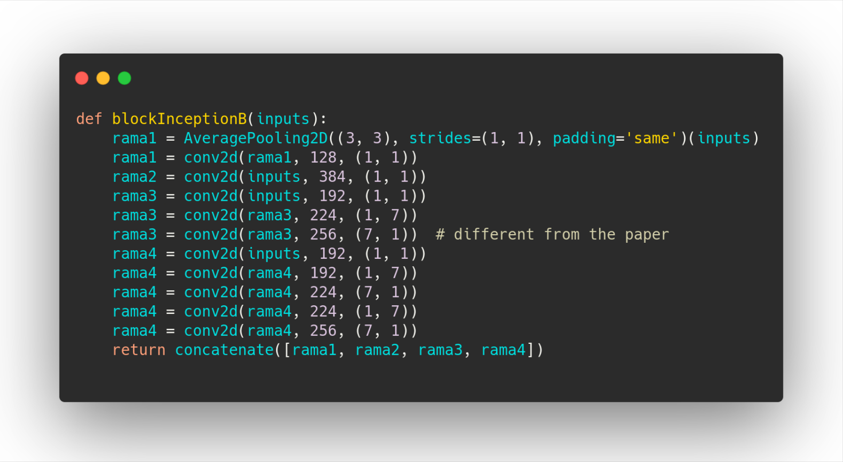
```python
def blockReductionA(inputs):
    rama1 = MaxPooling2D((3, 3), strides=(2, 2), padding='valid')(inputs)

    rama2 = conv2d(inputs, 384, (3, 3), strides=(2, 2), padding='valid')

    rama3 = conv2d(inputs, 192, (1, 1))
    rama3 = conv2d(rama3, 224, (3, 3))
    rama3 = conv2d(rama3, 256, (3, 3), strides=(2, 2), padding='valid')

    return concatenate([rama1, rama2, rama3])
```

Block Inception B



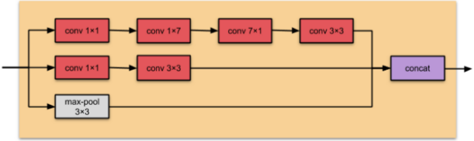
Inception-B

Block Inception B en keras



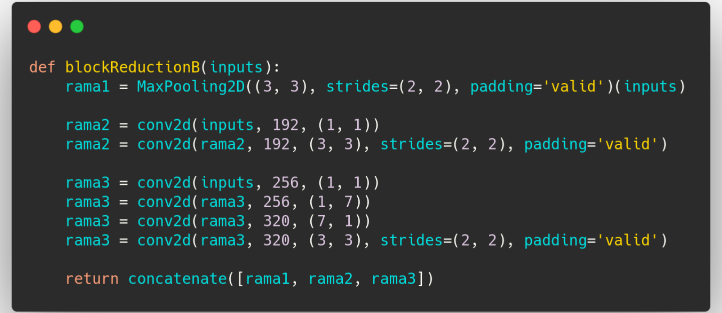
```
def blockInceptionB(inputs):
    rama1 = AveragePooling2D((3, 3), strides=(1, 1), padding='same')(inputs)
    rama1 = conv2d(rama1, 128, (1, 1))
    rama2 = conv2d(inputs, 384, (1, 1))
    rama3 = conv2d(inputs, 192, (1, 1))
    rama3 = conv2d(rama3, 224, (1, 7))
    rama3 = conv2d(rama3, 256, (7, 1))  # different from the paper
    rama4 = conv2d(inputs, 192, (1, 1))
    rama4 = conv2d(rama4, 192, (1, 7))
    rama4 = conv2d(rama4, 224, (7, 1))
    rama4 = conv2d(rama4, 224, (1, 7))
    rama4 = conv2d(rama4, 256, (7, 1))
    return concatenate([rama1, rama2, rama3, rama4])
```

Block Reduction B



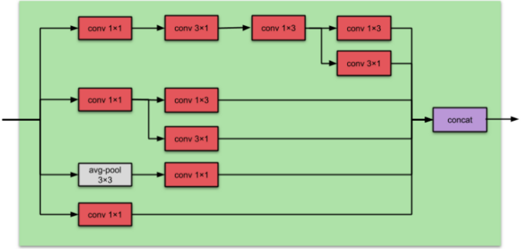
Reduction-B

Block Reduction B en keras



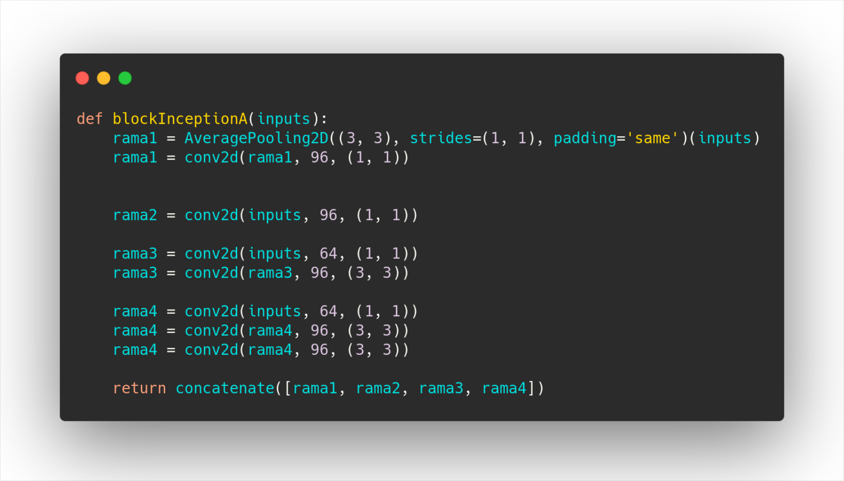
```
def blockReductionB(inputs):
    rama1 = MaxPooling2D((3, 3), strides=(2, 2), padding='valid')(inputs)

    rama2 = conv2d(inputs, 192, (1, 1))
    rama2 = conv2d(rama2, 192, (3, 3), strides=(2, 2), padding='valid')

    rama3 = conv2d(inputs, 256, (1, 1))
    rama3 = conv2d(rama3, 256, (1, 7))
    rama3 = conv2d(rama3, 320, (7, 1))
    rama3 = conv2d(rama3, 320, (3, 3), strides=(2, 2), padding='valid')

    return concatenate([rama1, rama2, rama3])
```

Block Inception C



Inception-C

Block Inception C en keras



```
def blockInceptionA(inputs):
    rama1 = AveragePooling2D((3, 3), strides=(1, 1), padding='same')(inputs)
    rama1 = conv2d(rama1, 96, (1, 1))


    rama2 = conv2d(inputs, 96, (1, 1))

    rama3 = conv2d(inputs, 64, (1, 1))
    rama3 = conv2d(rama3, 96, (3, 3))

    rama4 = conv2d(inputs, 64, (1, 1))
    rama4 = conv2d(rama4, 96, (3, 3))
    rama4 = conv2d(rama4, 96, (3, 3))

    return concatenate([rama1, rama2, rama3, rama4])
```

Construcción de la arquitectura

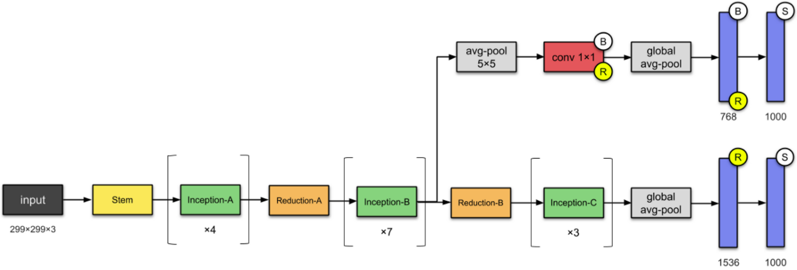
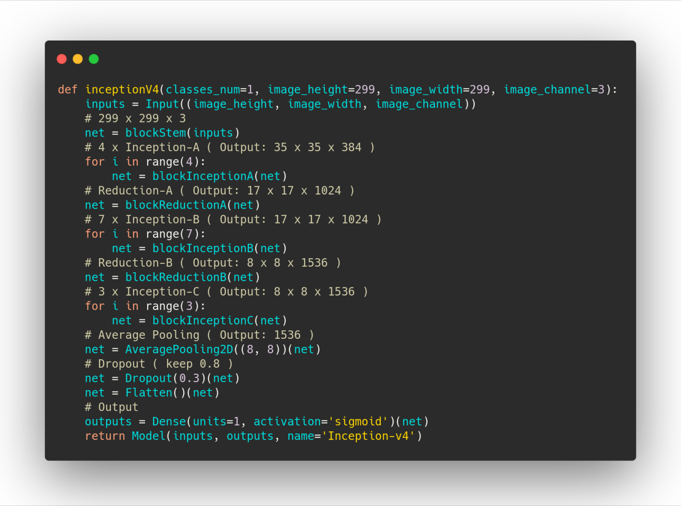


Figure: Modelo General

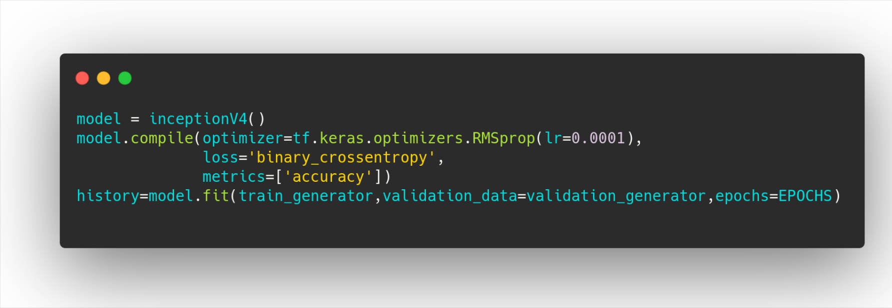
Construcción de la arquitectura en keras



```python
def inceptionV4(classes_num=1, image_height=299, image_width=299, image_channel=3):
    inputs = Input((image_height, image_width, image_channel))
    # 299 x 299 x 3
    net = blockStem(inputs)
    # 4 x Inception-A ( Output: 35 x 35 x 384 )
    for i in range(4):
        net = blockInceptionA(net)
    # Reduction-A ( Output: 17 x 17 x 1024 )
    net = blockReductionA(net)
    # 7 x Inception-B ( Output: 17 x 17 x 1024 )
    for i in range(7):
        net = blockInceptionB(net)
    # Reduction-B ( Output: 8 x 8 x 1536 )
    net = blockReductionB(net)
    # 3 x Inception-C ( Output: 8 x 8 x 1536 )
    for i in range(3):
        net = blockInceptionC(net)
    # Average Pooling ( Output: 1536 )
    net = AveragePooling2D((8, 8))(net)
    # Dropout ( keep 0.8 )
    net = Dropout(0.3)(net)
    net = Flatten()(net)
    # Output
    outputs = Dense(units=1, activation='sigmoid')(net)
    return Model(inputs, outputs, name='Inception-v4')
```

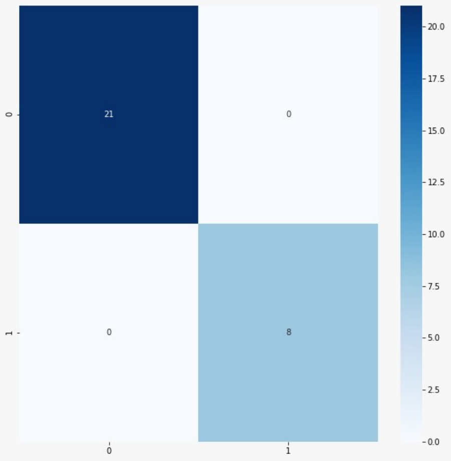
Figure: Inception v4

Compilación y entrenamiento

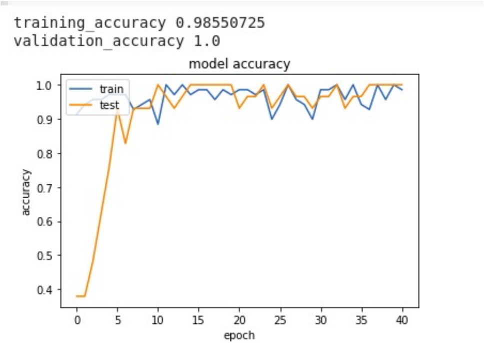


```
model = inceptionV4()
model.compile(optimizer=tf.keras.optimizers.RMSprop(lr=0.0001),
              loss='binary_crossentropy',
              metrics=['accuracy'])
history=model.fit(train_generator,validation_data=validation_generator,epochs=EPOCHS)
```

Resultados



Resultados



```
training_accuracy 0.98550725
validation_accuracy 1.0
```