Trabalho de Graduação

APLICAÇÃO DE REDE NEURAL PARA COMPRESSÃO DOS DADOS DO SATÉLITE GOES16

Ana Cristina de Paula Lima

ORIENTADOR: Prof.º Me. Emanuel Mineda Carneiro

COORIENTADOR: Dr.º Alex Sandro Aguiar Pessoa

Sumário

- Introdução
- Objetivo
- Fundamentação Teórica
- Desenvolvimento do trabalho
- Resultados e discussões
- Considerações Finais
- Referências

- Introdução
- Objetivo
- Fundamentação Teórica
- Desenvolvimento do trabalho
- Resultados e discussões
- Considerações Finais
- Referências

Introdução

Climatempo

- Empresa de consultoria meteorológica com sede em São Paulo e extensão em São José dos Campos.

GOES16

- Satélite que captura informações da América do Sul, através dos seus canais infravermelho, visível e vapor d’água.

GEONETCast

- Rede mundial que provém informações meteorológicas. As informações são obtidas a cada 15 minutos ou meia hora.

Dados observados

- São dados brutos obtidos dos satélites, salvos em formato NetCDF.

Dados de previsão

- Dados alterados pelos meteorologistas, que podem ser salvos em formato PNG.

Redes Neurais, Python, Keras

- Aplicação de rede neural do tipo *Autoencoder* através da linguagem *Python* e sua API denominada *Keras* para compressão das imagens em formato PNG.

- Introdução
- Objetivo
- Fundamentação Teórica
- Desenvolvimento do trabalho
- Resultados e discussões
- Considerações Finais
- Referências

Objetivo

Realizar a compressão das imagens geradas pela Climatempo, após a obtenção e tratamento dos dados via satélite GOES16, utilizando redes neurais.

Objetivos específicos

- Obter informações sobre os dados obtidos do satélite GOES16;
- Obter informações sobre as imagens geradas a partir dos dados de previsão;
- Realizar um estudo sobre redes neurais e a sua aplicação na resolução do problema;

- Realizar um estudo sobre as tecnologias que melhor se adequam à resolução do problema;
- Através dos resultados obtidos, comprovar a viabilidade em se utilizar redes neurais para este fim.

- Introdução
- Objetivo
- Fundamentação Teórica
- Desenvolvimento do trabalho
- Resultados e discussões
- Considerações Finais
- Referências

Fundamentação Teórica

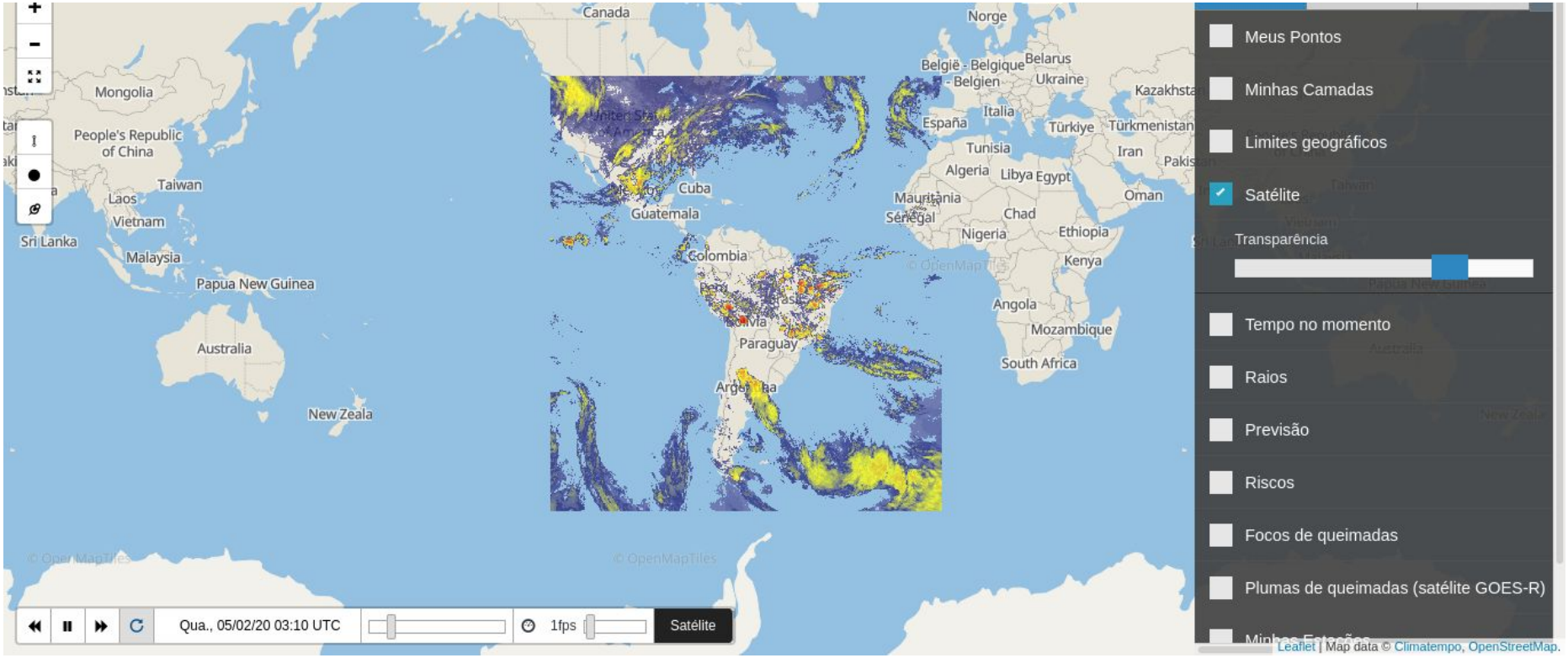
Climatempo

Empresa brasileira que oferece serviços e consultoria na área de meteorologia. Possui diversos produtos que utilizam informações advindas de satélites.

SMAC

O Sistema Meteorológico de Alertas Climatempo é utilizado para disparar alertas de raios e fenômenos meteorológicos, e disponibiliza as imagens de satélite em suas previsões.

Camada de satélite no SMAC



Fonte: (Autor, 2019)

GOES16

Primeiro satélite com detector de raios, possui cinco canais: um visível, três infravermelhos e um de vapor d'água. A sua posição na órbita favorece a América do Sul. (EMBRAPA, 2019)

GEONETCast

Rede mundial de sistemas que dissemina informações de satélite em tempo quase real a um baixo custo. (CPTEC - INPE, 2018)

Canal visível

Possui maior resolução espacial e a radiação solar é refletida pela superfície da atmosfera, medições noturnas são mais escuras. (YAMASOE, 2012)

Canal infravermelho

A radiação é emitida pela superfície e pela atmosfera e há medições no período noturno. Utilizado para medir propriedades térmicas da Terra. (FERNANDES, 2010)

Canal de vapor d’água

Utilizado para medir a quantidade de umidade presente na atmosfera, há disponibilidade de imagem em qualquer horário do dia. (YAMASOE, 2012)

NetCDF

Este formato possibilita o armazenamento e manipulação de dados científicos multidimensionais, tais como: temperatura, umidade, pressão, etc. (ARCGIS, 2019)

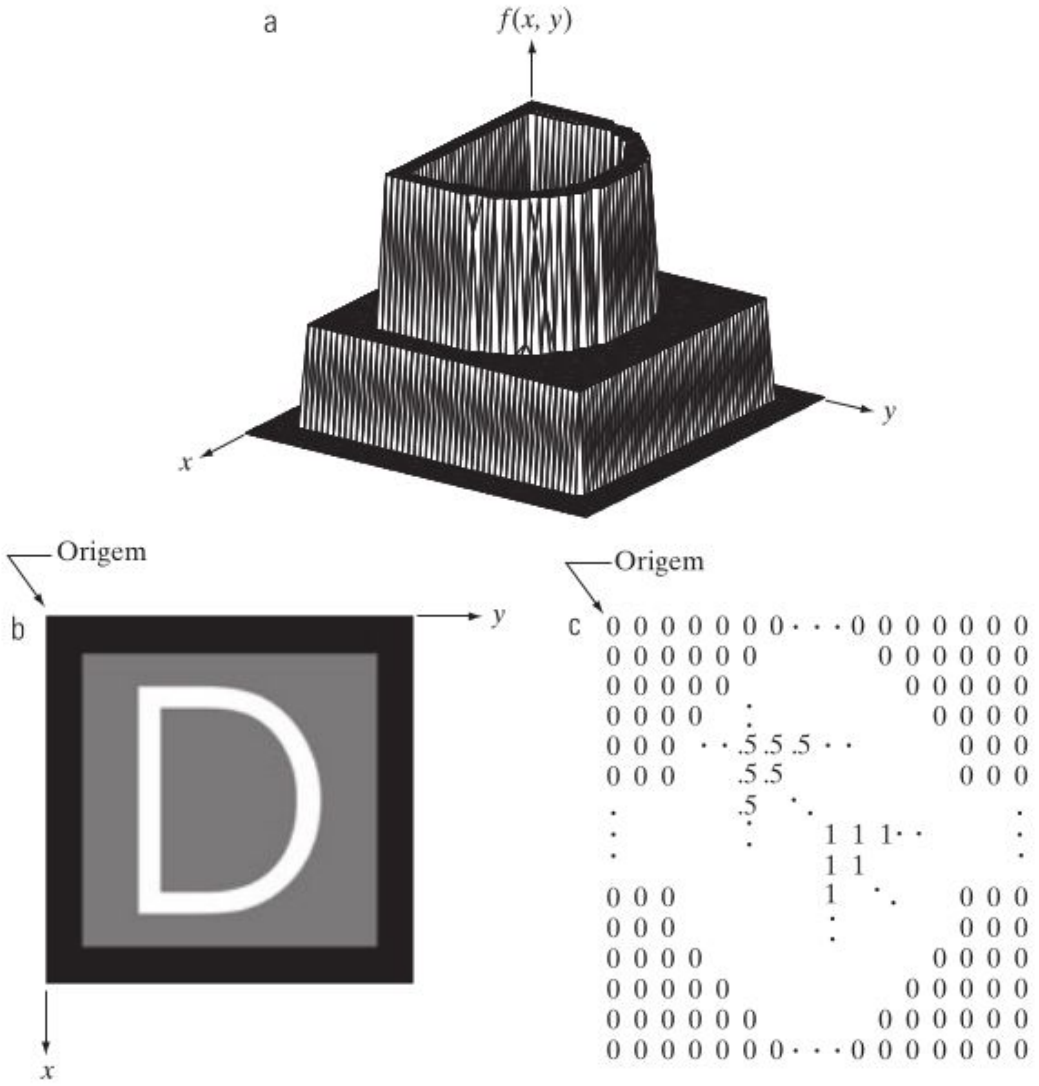
PNG

Este formato fornece arquivos de imagens de fácil portabilidade, com boa compressão e descompressão, flexibilidade e robustez. (W3C, 2019)

Processamento de imagem

Uma imagem digital é formada por um número finito de elementos, com localização e valor específicos, conhecidos como *Pixels.* (GONZALEZ; WOODS, 2010)

Representações de uma imagem digital

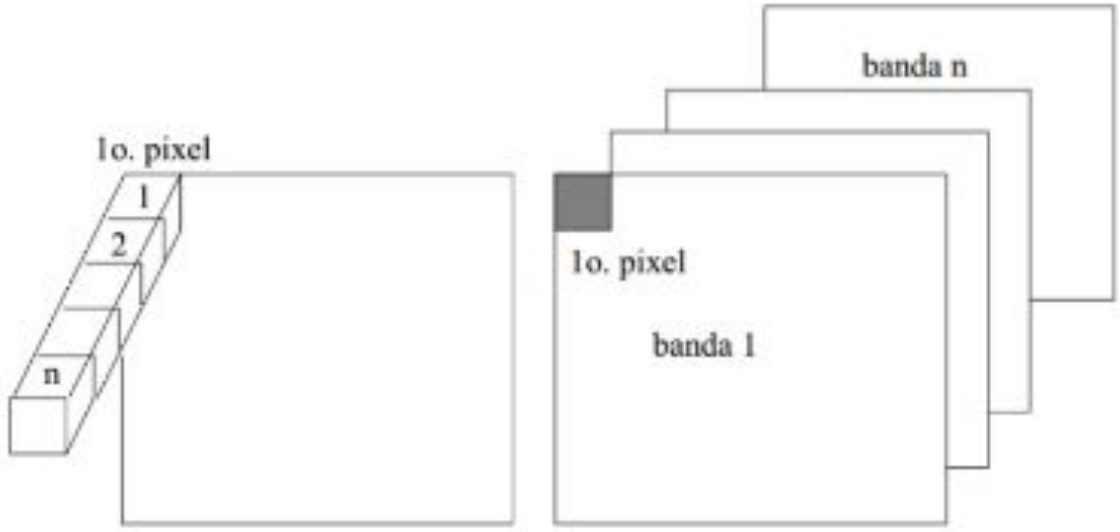


Fonte: (GONZALEZ; WOODS, 2010. p. 36)

Processamento de imagem colorida

A imagem colorida é considerada uma imagem multibanda, onde cada *pixel* representa uma combinação das cores primárias vermelha, verde e azul (RGB). (MARTINS, 2016)

Representação de uma imagem Multibanda

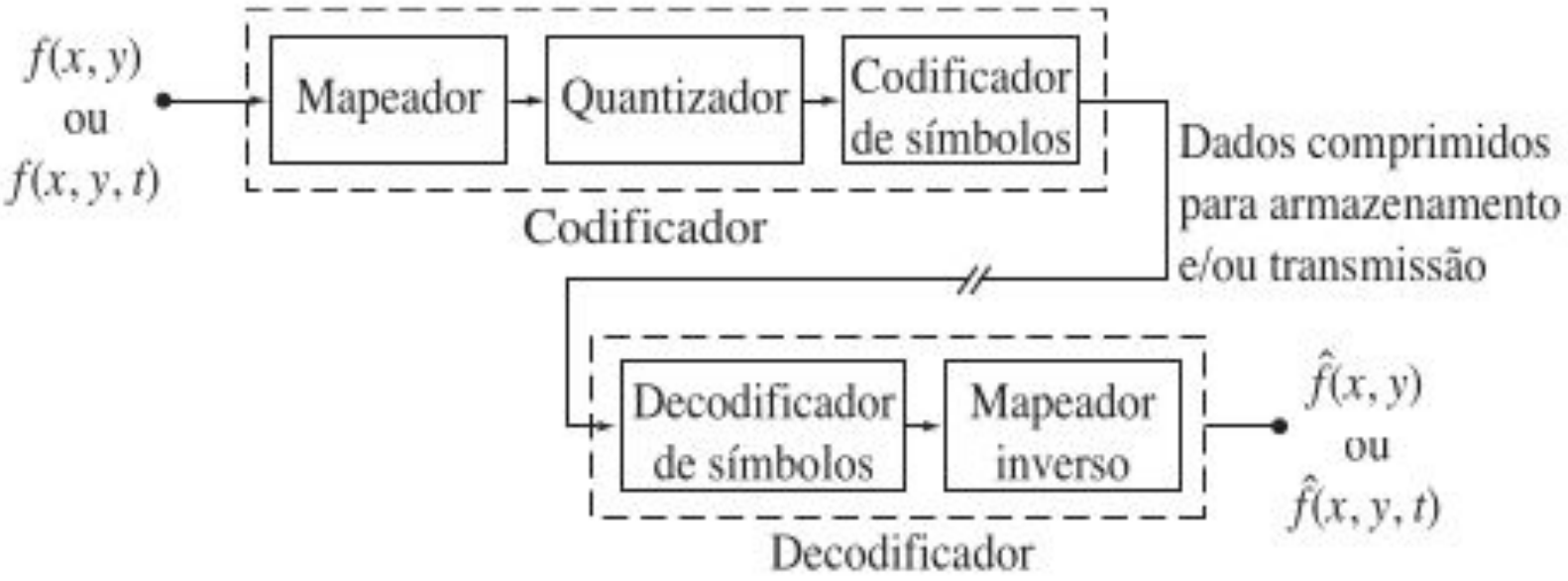


Fonte: (MARTINS, 2016)

Compressão de imagem

A compressão de imagens se refere ao processo de reduzir o volume de dados necessários para representar informações. Em imagens coloridas a compressão é aplicada separadamente em cada uma das bandas. (GONZALEZ; WOODS, 2010)

Diagrama de um sistema geral de compressão de imagens



Fonte: (GONZALEZ; WOODS, 2010. p. 357)

Redes Neurais Artificiais

Modelos computacionais baseados no sistema nervoso dos seres vivos. Possui camada de entrada, camadas ocultas de aprendizagem e camada de saída com os resultados. Utiliza dados supervisionados ou não supervisionados. (SILVA; SPATTI; FLAUZINO, 2010)

Autoencoders

Rede neural não supervisionada que é treinada para definir valores de saída iguais aos de entrada. (GOODFELLOW; BENGIO; COURVILLE, 2016)

Quando treinadas com um tipo específico de dados se tornam restritas à eles ou aos dados considerados semelhantes. (PURKAIT, 2019)

Deep Autoencoder

Uma rede neural do tipo *Deep Autoencoder* possui camadas ocultas interligadas. Modelos de redes profundas apresentam melhor aprendizagem. (GOODFELLOW; BENGIO; COURVILLE, 2016)

Função de ativação ReLU

A função de ativação ReLU é do tipo não linear e não ativa todos os neurônios ao mesmo tempo. A saída é o valor máximo entre 0 e o valor de entrada. (GOODFELLOW; BENGIO; COURVILLE, 2016)

Função de ativação Sigmóide

A função de ativação Sigmóide é do tipo não linear e é utilizada para prever probabilidades. Sua saída possui valores que variam entre 0 e 1. (GOODFELLOW; BENGIO; COURVILLE, 2016)

Função de otimização Adam

A função de otimização Adam calcula individualmente a taxa de aprendizado dos parâmetros estimados durante o treinamento.

(KHANDELWAL, 2019)

Função de perda RMSE

É possível calcular a perda da predição da camada de saída para com a camada de entrada. Lida com dados não supervisionados.
(SRINIVASAN, 2016)

Python, Tensorflow, Keras

Keras é uma API baseada em *Python* que pode ser utilizada em diferentes *backends*, no trabalho atual é utilizado o backend *Tensorflow*.

- Introdução
- Objetivo
- Fundamentação Teórica
- Desenvolvimento do trabalho
- Resultados e discussões
- Considerações Finais
- Referências

Desenvolvimento do trabalho

Para a realização deste trabalho foi criada uma classe denominada *Deep Autoencoder,* que reduzirá as imagens para as seguintes dimensões: 512, 384, 256, 128 e 64 *pixels.*

Através dessa classe será possível receber os dados de entrada, treinar os modelos e gerar as imagens compressadas e descompressadas.

Ao instanciar a classe *DeepAutoencoder* é preciso passar como parâmetros o valor de entrada e de encodificação.

```
class DeepAutoencoder(object):
  def __init__(self, input_dim, encoded_dim):
    # Dimensions of output and input layers
    input_layer = Input(shape=(input_dim,))
    hidden_input = Input(shape=(encoded_dim,))
```

Nesse trecho ocorre a criação das camadas responsáveis pela encodificação:

```
# Hidden layers to encoder (512, 384, 256, 128)
encoded = Dense(8 * encoded_dim, activation='relu')(input_layer)
encoded = Dense(6 * encoded_dim, activation='relu')(encoded)
encoded = Dense(4 * encoded_dim, activation='relu')(encoded)
encoded = Dense(2 * encoded_dim, activation='relu')(encoded)
```

Essa é a menor camada criada a partir do valor dimensional de encodificação passado inicialmente:

```
# Hidden layer (64)
hidden_layer = Dense(encoded_dim, activation='relu')(encoded)
```

Nesse trecho ocorre a criação das camadas responsáveis pela decodificação:

```
# Hidden layers to decoder (128, 256, 384, 512)
decoded = Dense(2 * encoded_dim, activation='relu')(hidden_layer)
decoded = Dense(4 * encoded_dim, activation='relu')(decoded)
decoded = Dense(6 * encoded_dim, activation='relu')(decoded)
decoded = Dense(8 * encoded_dim, activation='relu')(decoded)
```

Essa é a camada de saída criada a partir do valor dimensional de entrada passado inicialmente:

```
# Output layer
output_layer = Dense(input_dim, activation='sigmoid')(decoded)
```

Estes são os modelos criados do *autoencoder* e do *encoder*:

```
# Autoencoder and encoder models
self.autoencoder = Model(input_layer, output_layer)
self.encoder = Model(input_layer, hidden_layer)
```

Sequência das camadas escondidas encodificadas:

```
# Sequential hidden layers of encoder
layer1 = self.autoencoder.layers[-5]
layer2 = self.autoencoder.layers[-4]
layer3 = self.autoencoder.layers[-3]
layer4 = self.autoencoder.layers[-2]
layer5 = self.autoencoder.layers[-1]
```

Reconstrução da camada de entrada a partir das camadas escondidas e criação do modelo *decoder*:

```
encoded_layers = layer5(layer4(layer3(layer2(layer1(hidden_input)))))

# Decoder model
self.decoder = Model(hidden_input, encoded_layers)
```

Compilação do modelo *autoencoder:*

```
# Compiler autoencoder using optimizer function adam
self.autoencoder.compile(optimizer='adam', loss=rmse)
```

Função responsável pelo treinamento do modelo:

```
# Methods
# Method train autoencoder
def train(self, input_train, input_test, batch_size, epochs):
  self.autoencoder.fit(input_train, input_train,
            epochs=epochs,
            batch_size=batch_size,
            shuffle=True,
            validation_data=(input_train, input_test))
```

Funções responsáveis pela recuperação das imagens encodificadas e decodificadas:

```
# Method to return an encoded image
def get_encoded_image(self, image):
  encoded_img = self.encoder.predict(image)
  return encoded_img


# Method to return a decoded image
def get_decoded_image(self, encoded_img):
  decoded_img = self.decoder.predict(encoded_img)
  return decoded_img
```

Normalização das imagens:

```
# Normalization of training image and test image
img_train = mpimg.imread((os.path.join(data_png, img)))
train = img_train.reshape((len(img_train), np.prod(img_train.shape[1:])))

img_test = mpimg.imread((os.path.join(data_png, img)))
test = img_train.reshape((len(img_train), np.prod(img_train.shape[1:])))
```

Utilização da classe *DeepAutoencoder* e a recuperação da camada encodificada:

```
# Instantiation of class DeepAutoencoder
deep_autoencoder = DeepAutoencoder(train.shape[1], 64)
deep_autoencoder.train(train, test, 64, 100)


# Get encoded and decoded image
encoded_img = deep_autoencoder.get_encoded_image(test)
decoded_img = deep_autoencoder.get_decoded_image(encoded_img)
```

Modelo Autoencoder da Imagem Infravermelho

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 26716)]	0
dense (Dense)	(None, 512)	13679104
dense_1 (Dense)	(None, 384)	196992
dense_2 (Dense)	(None, 256)	98560
dense_3 (Dense)	(None, 128)	32896
dense_4 (Dense)	(None, 64)	8256
dense_5 (Dense)	(None, 128)	8320
dense_6 (Dense)	(None, 256)	33024
dense_7 (Dense)	(None, 384)	98688
dense_8 (Dense)	(None, 512)	197120
dense_9 (Dense)	(None, 26716)	13705308

Total params: 28,058,268
Trainable params: 28,058,268
Non-trainable params: 0

Fonte: (Autor, 2019)

Modelo Encoder da Imagem Infravermelho

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 26716)]           0
_________________________________________________________________
dense (Dense)                (None, 512)               13679104
_________________________________________________________________
dense_1 (Dense)              (None, 384)               196992
_________________________________________________________________
dense_2 (Dense)              (None, 256)               98560
_________________________________________________________________
dense_3 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_4 (Dense)              (None, 64)                8256
=================================================================
Total params: 14,015,808
Trainable params: 14,015,808
Non-trainable params: 0
```

Fonte: (Autor, 2019)

Modelo Decoder da Imagem Infravermelho

```
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 64)]              0
_________________________________________________________________
dense_5 (Dense)              (None, 128)               8320
_________________________________________________________________
dense_6 (Dense)              (None, 256)               33024
_________________________________________________________________
dense_7 (Dense)              (None, 384)               98688
_________________________________________________________________
dense_8 (Dense)              (None, 512)               197120
_________________________________________________________________
dense_9 (Dense)              (None, 26716)             13705308
=================================================================
Total params: 14,042,460
Trainable params: 14,042,460
Non-trainable params: 0
```

Fonte: (Autor, 2019)

Dez primeiras épocas do treino da Imagem Infravermelho

```
Train on 6122 samples, validate on 6122 samples
Epoch 1/100
6122/6122 [==============================] - 3s 482us/sample - loss: 0.1759 - val_loss: 0.1548
Epoch 2/100
6122/6122 [==============================] - 2s 371us/sample - loss: 0.1511 - val_loss: 0.1461
Epoch 3/100
6122/6122 [==============================] - 2s 363us/sample - loss: 0.1420 - val_loss: 0.1365
Epoch 4/100
6122/6122 [==============================] - 2s 368us/sample - loss: 0.1328 - val_loss: 0.1338
Epoch 5/100
6122/6122 [==============================] - 2s 360us/sample - loss: 0.1271 - val_loss: 0.1196
Epoch 6/100
6122/6122 [==============================] - 2s 355us/sample - loss: 0.1158 - val_loss: 0.1095
Epoch 7/100
6122/6122 [==============================] - 2s 365us/sample - loss: 0.1122 - val_loss: 0.1049
Epoch 8/100
6122/6122 [==============================] - 2s 372us/sample - loss: 0.1043 - val_loss: 0.1028
Epoch 9/100
6122/6122 [==============================] - 2s 362us/sample - loss: 0.1012 - val_loss: 0.0994
Epoch 10/100
6122/6122 [==============================] - 2s 364us/sample - loss: 0.0980 - val_loss: 0.0974
```

Fonte: (Autor, 2019)

Dez últimas épocas do treino da Imagem Infravermelho

```
Epoch 90/100
6122/6122 [==============================] - 2s 370us/sample - loss: 0.0632 - val_loss: 0.0617
Epoch 91/100
6122/6122 [==============================] - 2s 366us/sample - loss: 0.0623 - val_loss: 0.0610
Epoch 92/100
6122/6122 [==============================] - 2s 360us/sample - loss: 0.0618 - val_loss: 0.0606
Epoch 93/100
6122/6122 [==============================] - 2s 357us/sample - loss: 0.0628 - val_loss: 0.0625
Epoch 94/100
6122/6122 [==============================] - 2s 367us/sample - loss: 0.0630 - val_loss: 0.0611
Epoch 95/100
6122/6122 [==============================] - 2s 369us/sample - loss: 0.0625 - val_loss: 0.0621
Epoch 96/100
6122/6122 [==============================] - 2s 366us/sample - loss: 0.0630 - val_loss: 0.0629
Epoch 97/100
6122/6122 [==============================] - 2s 375us/sample - loss: 0.0627 - val_loss: 0.0613
Epoch 98/100
6122/6122 [==============================] - 2s 367us/sample - loss: 0.0626 - val_loss: 0.0609
Epoch 99/100
6122/6122 [==============================] - 2s 373us/sample - loss: 0.0627 - val_loss: 0.0618
Epoch 100/100
6122/6122 [==============================] - 2s 368us/sample - loss: 0.0639 - val_loss: 0.0630
```

Fonte: (Autor, 2019)

- Introdução
- Objetivo
- Fundamentação Teórica
- Desenvolvimento do trabalho
- Resultados e discussões
- Considerações Finais
- Referências

Resultados e discussões

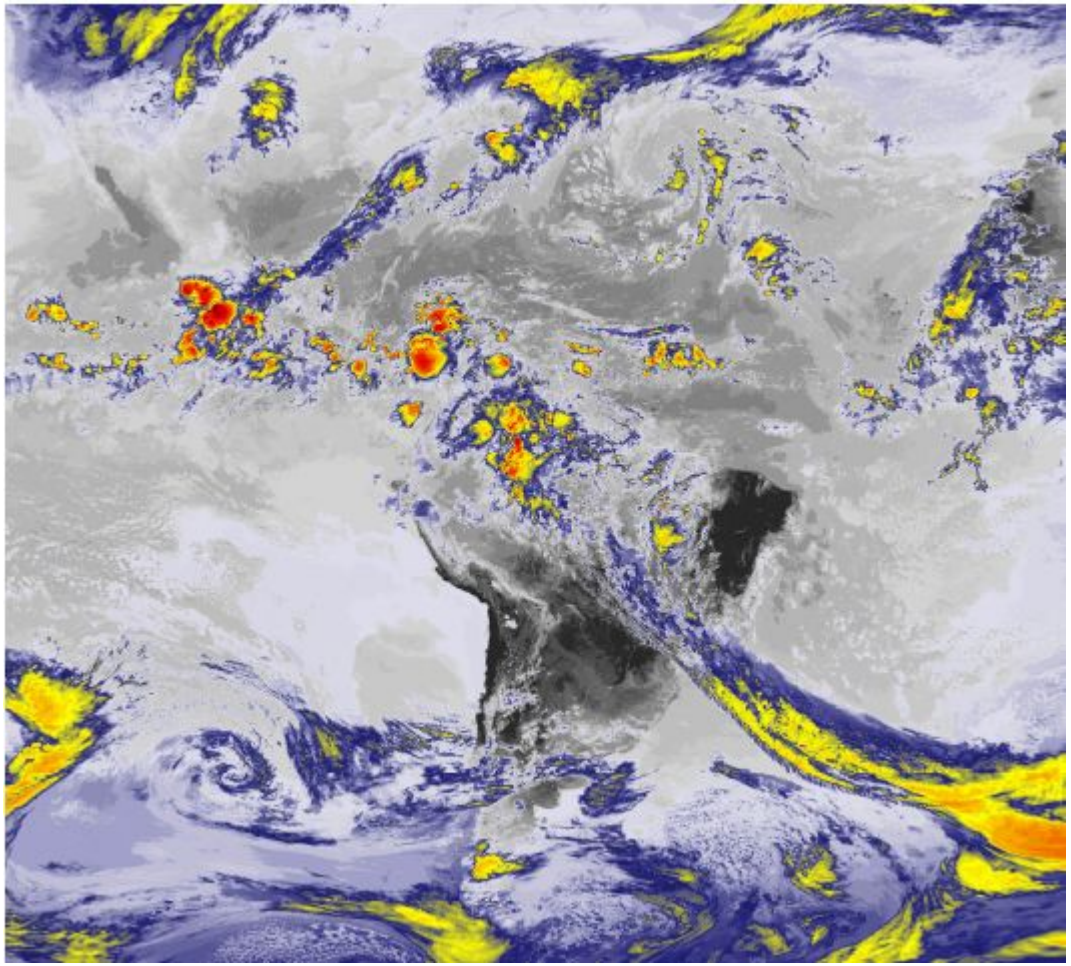
Os resultados demonstrados a seguir são referentes à imagem infravermelho.

Informações da Imagem Infravermelho

```
Image:  GOESR_RET_CH13_IRCOL0_20191009_1400.png
Dimensão inicial:  3
Shape inicial: (6122, 6679, 4)
Tamanho inicial:  163555352
Dimensão normalizada:  2
Shape normalizado: (6122, 26716)
Tamanho normalizado:  163555352
```

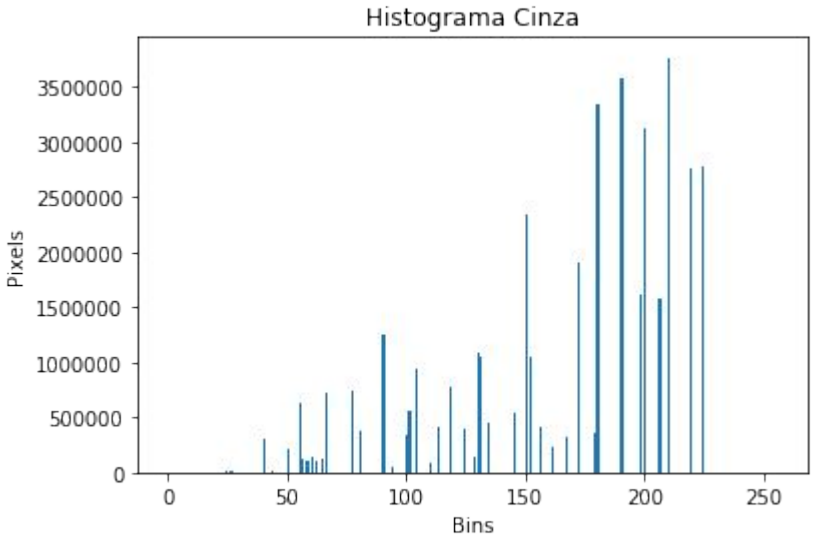
Fonte: (Autor, 2019)

Imagem Infravermelho de entrada



Fonte: (Autor, 2019)

Histograma cinza e colorido da imagem Infravermelho de entrada



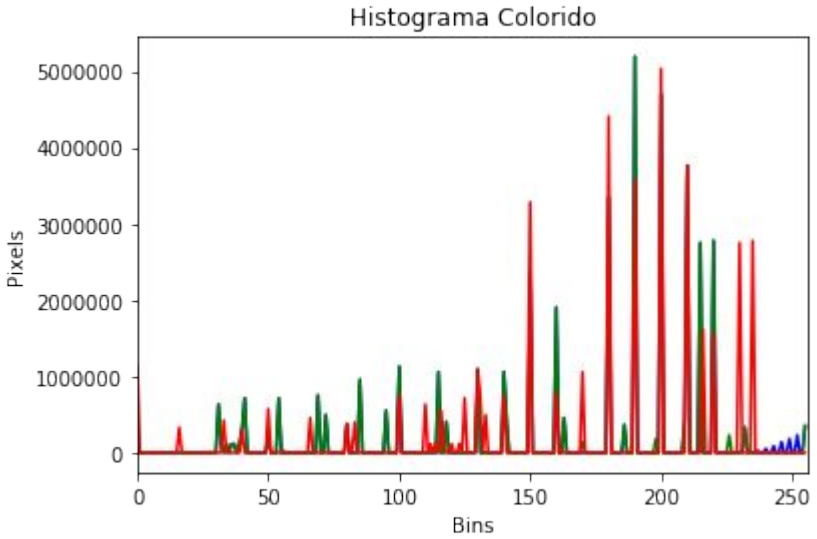
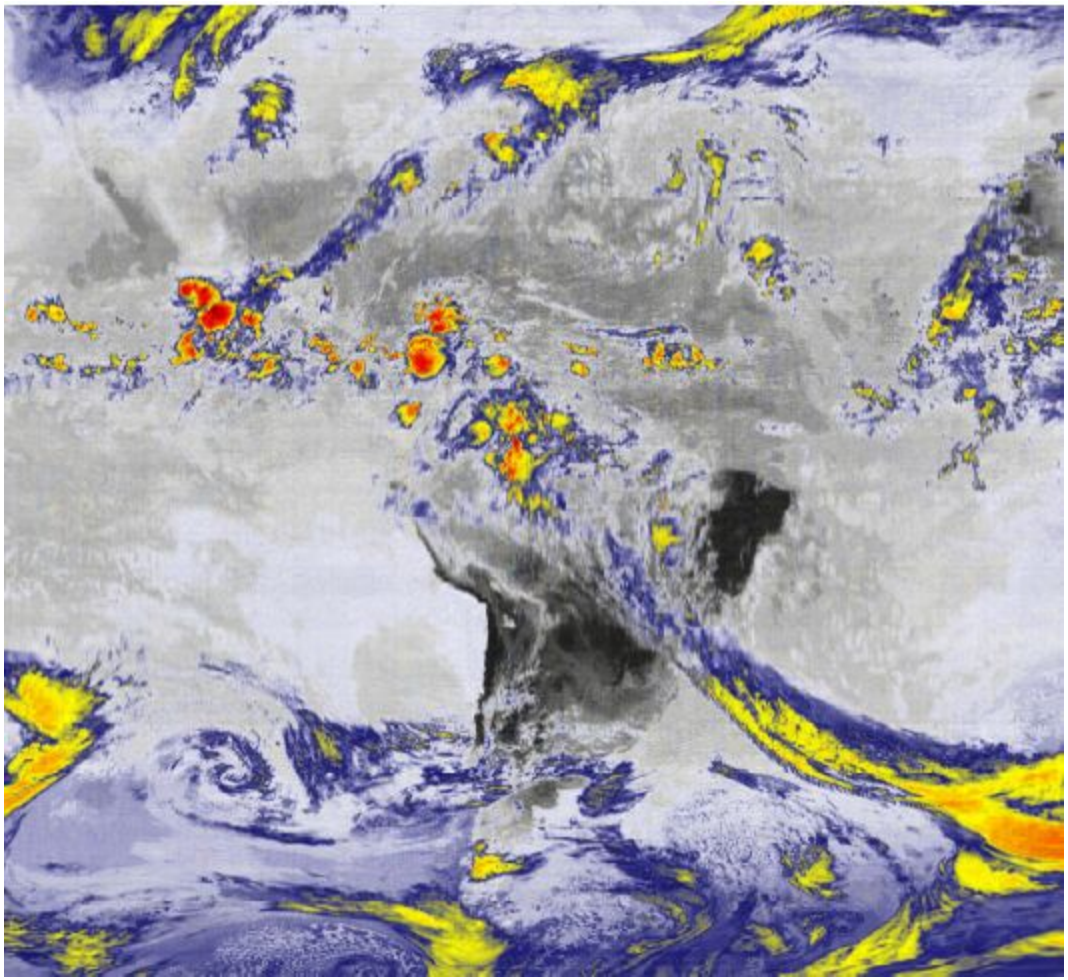
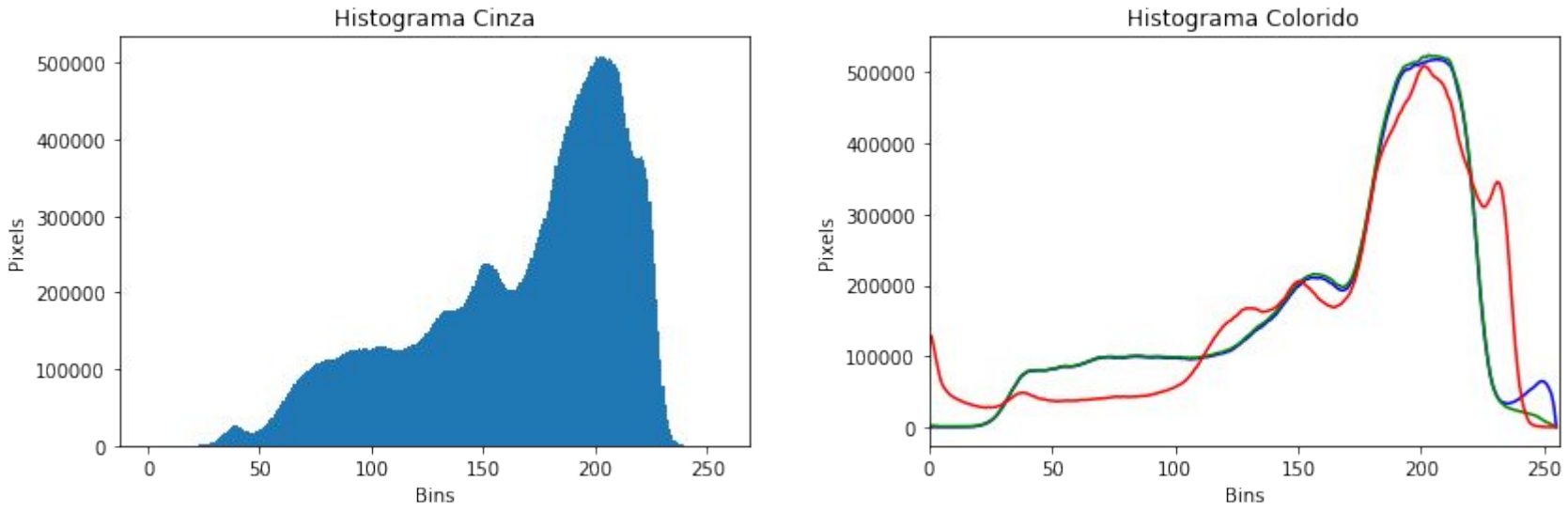


Imagem Infravermelho de saída



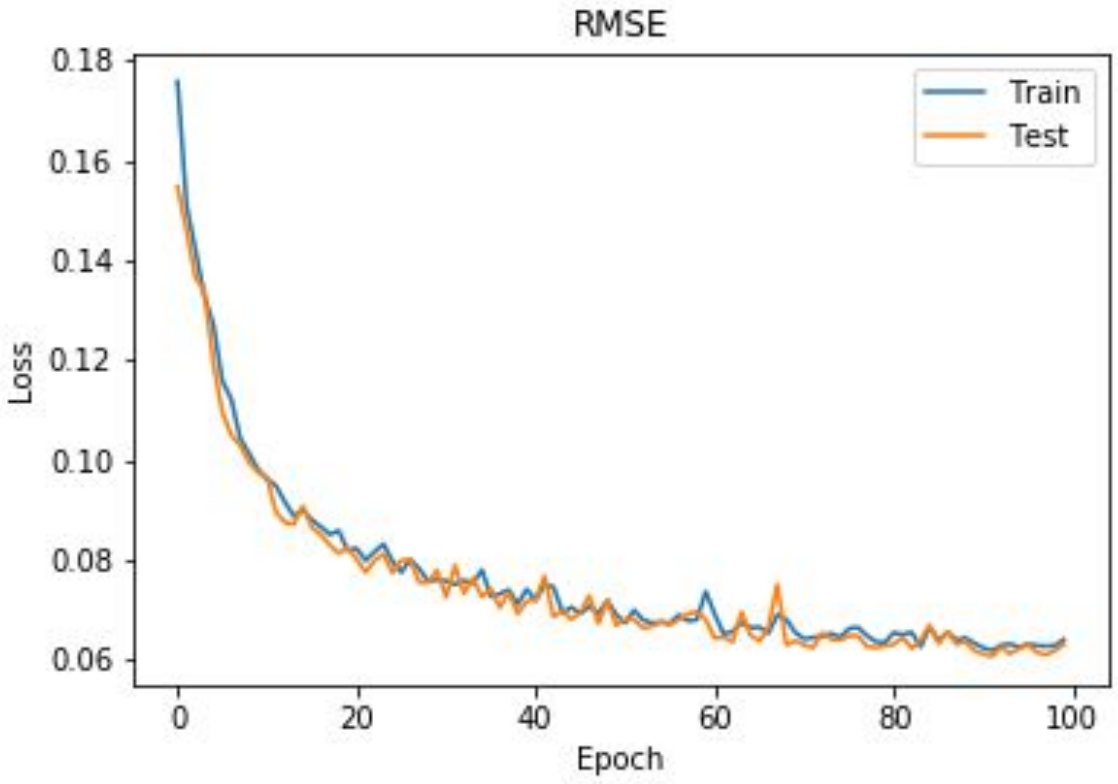
Fonte: (Autor, 2019)

Histograma cinza e colorido da imagem Infravermelho de saída



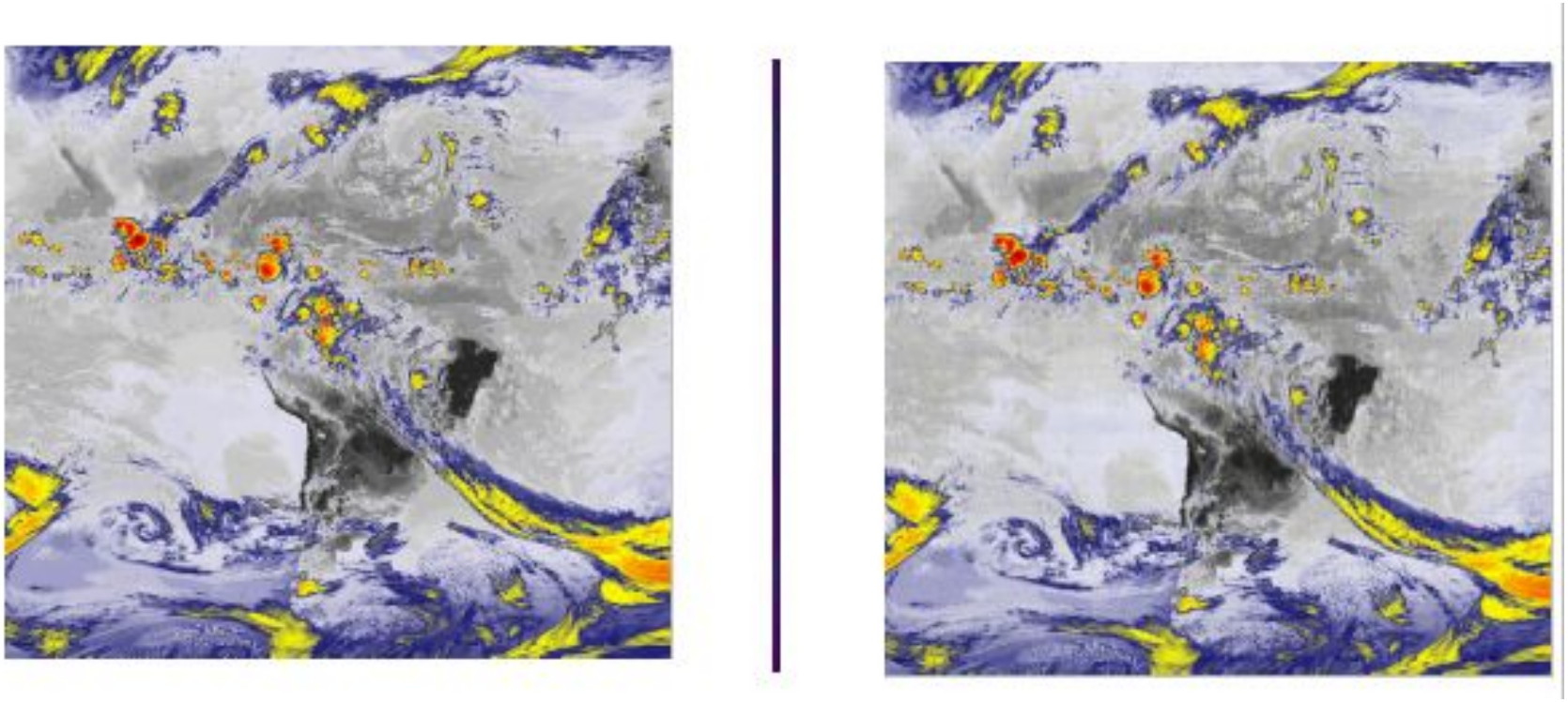
Fonte: (Autor, 2019)

Gráfico de perda da imagem Infravermelho



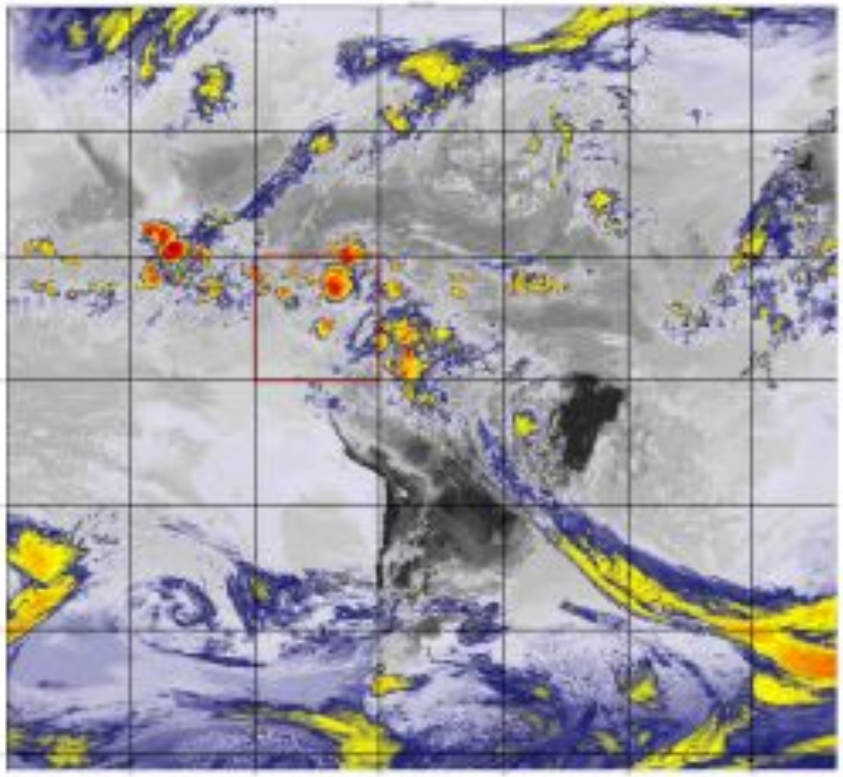
Fonte: (Autor, 2019)

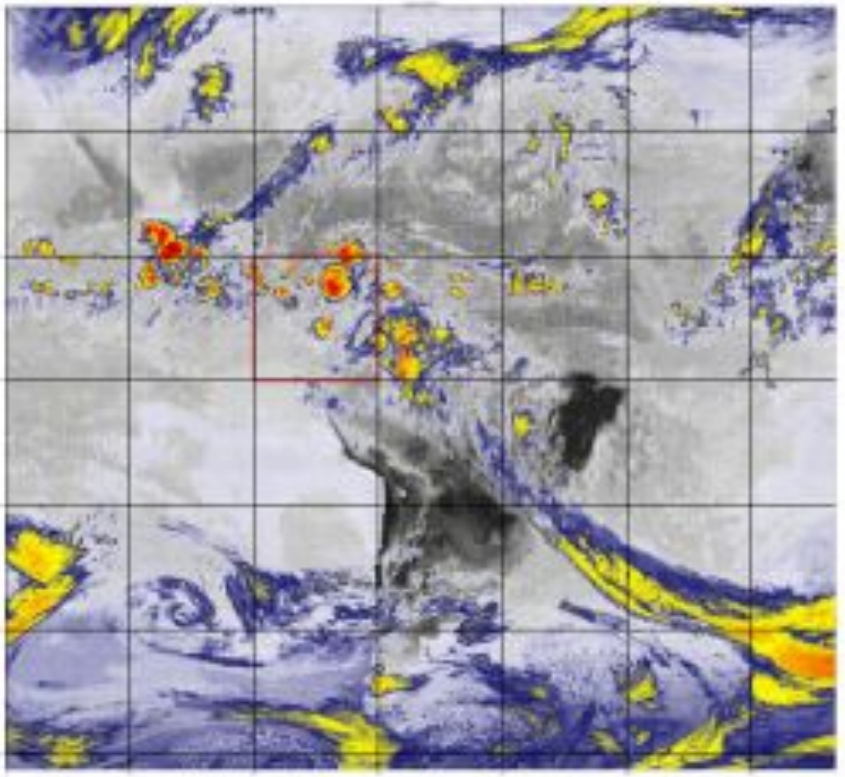
Imagens Infravermelho



Fonte: (Autor, 2019)

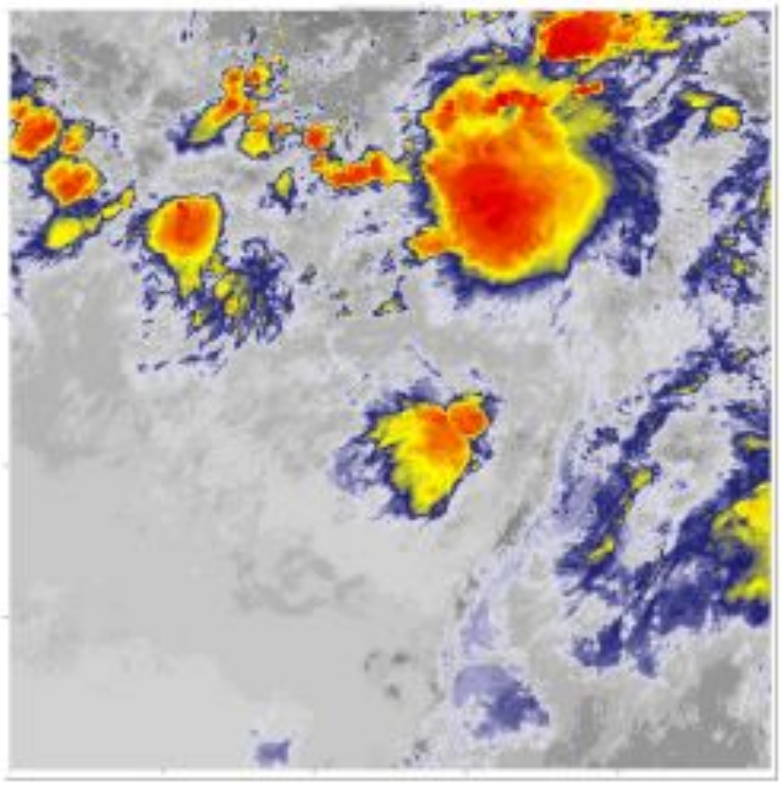
Primeira área selecionada da imagem Infravermelho

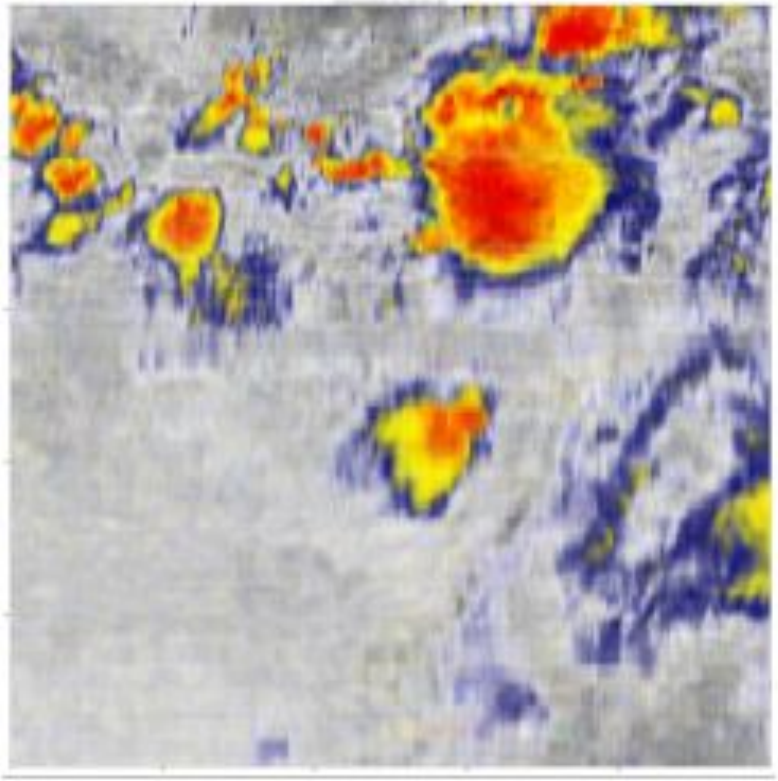




Fonte: (Autor, 2019)

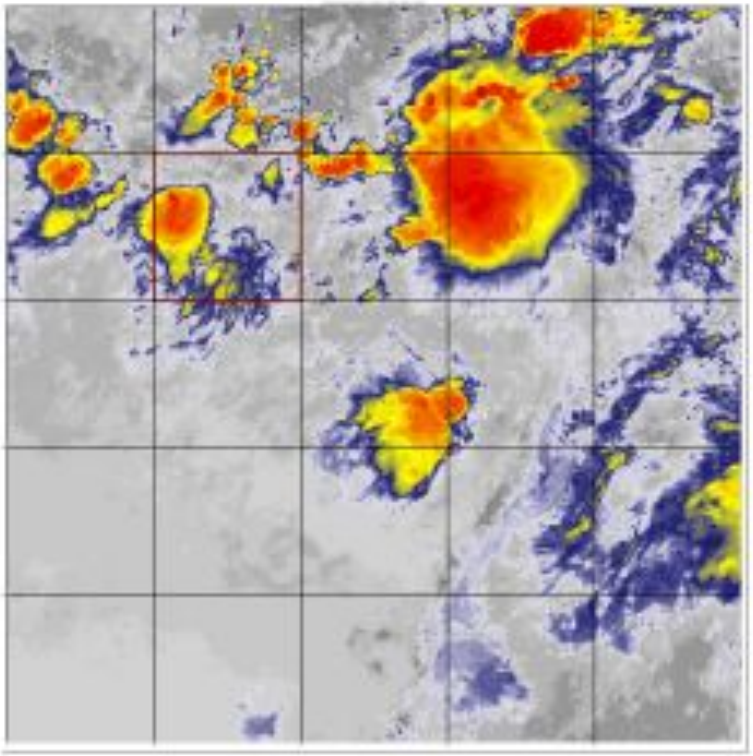
Primeiro recorte da imagem Infravermelho

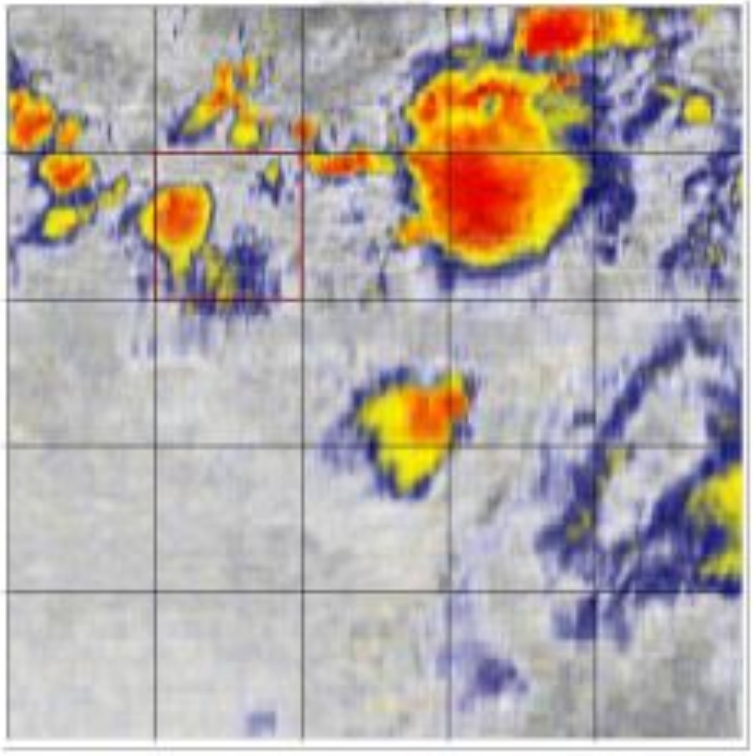




Fonte: (Autor, 2019)

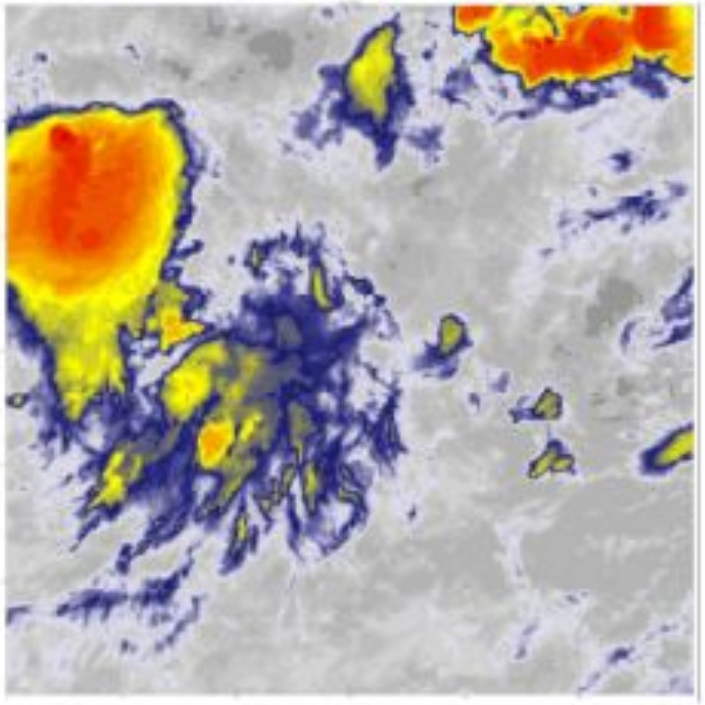
Segunda área selecionada da imagem Infravermelho

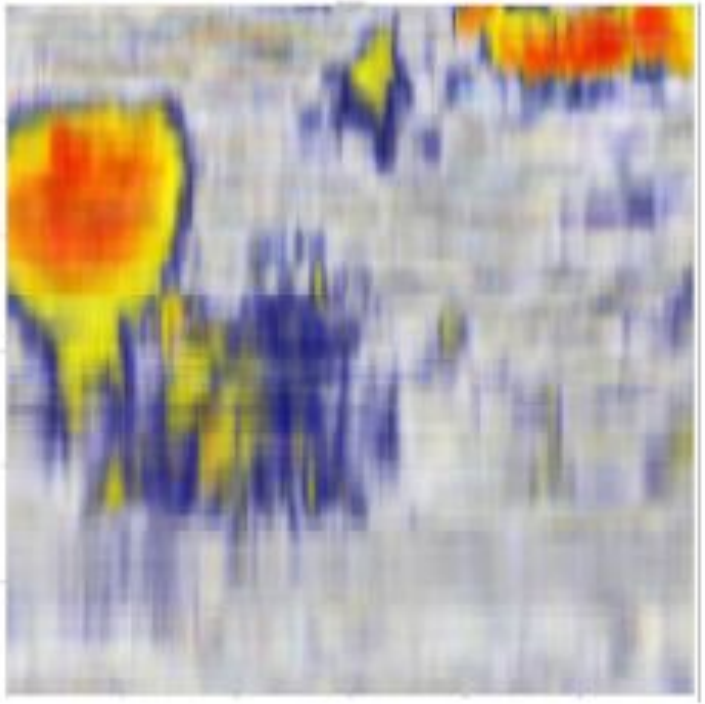




Fonte: (Autor, 2019)

Segundo recorte da imagem Infravermelho





Fonte: (Autor, 2019)

- Introdução
- Objetivo
- Fundamentação Teórica
- Desenvolvimento do trabalho
- Resultados e discussões
- Considerações Finais
- Referências

Considerações Finais

A imagem de satélite após a compressão apresentou um bom resultado, porém, com a aplicação de *zoom* nas imagens, é possível notar que a qualidade não é a mesma da imagem original.

Contribuições

As contribuições oferecidas por este trabalho foram:

- Criação de rede neural do tipo *Deep Autoencoder* para compressão e descompressão de imagens de satélite;
- Aplicação da *API* do *Keras* para criação de redes neurais;
- Aplicação da função de otimização *Adam*, funções de ativação *ReLU* e Sigmóide e o uso da função de perda *RMSE;*
- Normalização dos dados de entrada de uma imagem de 3 dimensões para 2 dimensões.

Trabalhos futuros

Apesar das contribuições apresentadas, existem algumas modificações e melhorias a serem realizadas. Abaixo algumas melhorias e sugestões que podem ser implementadas:

- Padronização das imagens em um tamanho único para a criação de um *dataset* que possa ser utilizado no treino dos modelos*;*
- Otimizar a criação dos modelos para que seja possível decodificar imagens similares àquelas utilizadas no treino;

- Criação de uma *API* que receba os dados encodificados e retorne a imagem decodificada, dessa forma seria necessário apenas armazenar os dados encodificados que ocupam menos espaço em disco;

- Combinar diferentes redes neurais do tipo *Autoencoder* para melhorar a qualidade das imagens decodificadas.

- Introdução
- Objetivo
- Fundamentação Teórica
- Desenvolvimento do trabalho
- Resultados e discussões
- Considerações Finais
- Referências

Referências

EMBRAPA. **GOES - Geostationary Operational Environmental Satellite.** Disponível em: <https://www.cnpm.embrapa.br/projetos/sat/conteudo/missao_goes.html>. Acesso em: 08 jun. 2019.

CPTEC - INPE. **GEONETCast Americas.** Disponível em: <http://satelite.cptec.inpe.br/geonetcast/br/>. Acesso em: 13 jun. 2018.

FERNANDES, Diego Simões. **UNIVERSIDADE DE SÃO PAULO INSTITUTO DE ASTRONOMIA, GEOFÍSICA E CIÊNCIAS ATMOSFÉRICAS DEPARTAMENTO DE CIÊNCIAS ATMOSFÉRICAS Caracterização das Tempestades a partir dos canais Infravermelho e Vapor d'água do Satélite GOES 10 e 12.** 2010. 165 f. TCC (Graduação) - Curso de Instituto de Astronomia, Geofísica e Ciências Atmosféricas departamento de Ciências Atmosféricas, Universidade de São Paulo, São Paulo, 2010. Disponível em: <http://www.iag.usp.br/pos/sites/default/files/m_diego_s_f.pdf>. Acesso em: 09 jun. 2019.

ARCGIS. **What is NetCDF?** 2019. Disponível em: <https://pro.arcgis.com/en/pro-app/help/data/multidimensional/what-is-netcdf-data.htm>. Acesso em: 04 jan. 2020.

W3C. **PNG.** 2019. Disponível em: <https://www.w3.org/TR/REC-png.pdf>. Acesso em: 04 jan. 2020.

GONZALEZ, Rafael C.; WOODS, Richard E.. **Processamento Digital de Imagens.** 3. ed. São Paulo: Pearson Education, 2010. 644 p.

MARTINS, Samuel Botter. **Introdução ao Processamento Digital de Imagens:** Parte 1 - Definições Básicas, Espaço de Cores e Histogramas. Campinas: Unicamp, 2016. 15 p.

SILVA, Ivan Nunes da; SPATTI, Danilo Hernane; FLAUZINO, Rogério Andrade. **Redes Neurais Artificiais:** Para engenharia e Ciências Aplicadas. São Paulo: Artliber, 2010. 395 p.

PURKAIT, Niloy. **Hands-On Neural Networks with Keras.** United Kingdom: Packt Publishing, 2019.

KHANDELWAL, Renu. **Deep Autoencoder using Keras.** 2019. Disponível em: <https://medium.com/datadriveninvestor/deep-autoencoder-using-keras-b77cd3e8be95>. Acesso em: 02 jan. 2020.

SRINIVASAN, Krishnan. **Autoencoders.** 2016. Disponível em: <https://yaledatascience.github.io/2016/10/29/autoencoders.html>. Acesso em: 29 out. 2016.

YAMASOE, Marcia. **Interpretação de Imagens:** São Paulo, 2012. 83 slides, color. Disponível em: <http://dca.iag.usp.br/material/akemi/satelite/Interpreta%E7%E3o%20de%20Imagens_2012.pdf>. Acesso em: 02 abr. 2019.

Muito Obrigada!

analimazn@gmail.com