# Nanodegree Engenheiro de Machine Learning

***Arthur Sena, 14/08/2018***

# Projeto Final - Relatório

## 1) Introdução

O cérebro humano é uma das partes mais fascinantes do corpo humano. Ele é o núcleo que controla diversas das nossas ações diárias. Desde às mais complexas, como resolver uma derivada numa prova de cálculo I, até às mais simples, como reconhecer se uma foto contém ou não um cachorro. E é sobre essa ação de reconhecer uma determinada imagem, foto ou retrato que esse projeto se trata. Essa que parece ser uma tarefa simples para nós, seres humanos, não é tão fácil para um computador que apenas enxerga números binários.

Por esse motivo que, ao longo das últimas décadas, diversos algoritmos e técnicas foram desenvolvidos com o intuito de indentificar e reconhecer padrões em imagens. Assim sendo, esse relatório descreve o desenvolvimento de um modelo de aprendizagem de máquina que utiliza tais técnicas e é capaz de reconhecer imagens de três tipos diferentes de ambientes:

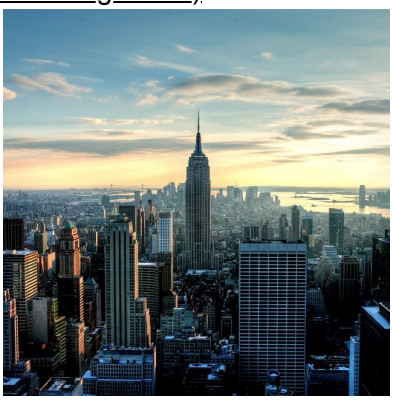
- Urbano: Imagens relacionadas com ambientes de cidades, prédios, casas, etc.
- Natureza: Imagens relacionadas com ambientes de florestas, matas, jardins, etc.
- Praia/Litoral: Imagens realacionadas com praias, mares, areia, etc.

É importante ressaltar que o foco desse projeto foi a utilização de algoritmos de *Deep Learning*, visto que estes apresentam os melhores resultados no campo de reconhecimento de imagem como um todo. Nas seções abaixos, é descrito todos os passos que envolveram o desenvolvimento do melhor modelo encontrado.

## 2) Dados

O primeiro desafio encontrado foi a necessidade de construção de uma base de dados de imagens que contivesse os ambientes que estamos tentando reconhecer. Cada um desses ambientes possue uma determinada peculiaridade, padrão e caracteristica que deve estar presentes na nossa base, pois assim o nosso modelo irá aprendê-lo. Dessa forma, resolvemos utilizar a rede social de fotos mais famosa do mundo conhecida como _Instagram_ como a fonte da nossa base. Para isso, foi criado um _script_ em python que acessou os seguintes perfis :

- Perfil Urbano (https://www.instagram.com/big.cities/)



Example 1

- Perfil de Praias e Litoral (https://www.instagram.com/beaches_n_resorts/)

Example 2

- Perfil de Natureza e floretas (https://www.instagram.com/forest/)

Example 3

O nosso script acessou e coletou cada uma das fotos dos links acima, resultando numa base com **945** imagens, sendo **259** fotos de cidades, **292** de florestas e **394** de praias. Todas essas imagens estão coloridas e apresentam um tamanho original de 640x640. Observando os exemplos de imagens acima, conseguimos facilmente distinguir cada um, onde o jogo de cores e os objetos inseridos nas imagens nos ajudam a fazer tal distinção. Portanto, o desafio aqui é ensinar para uma máquina como ela pode usar tais características a fim de discriminar as fotos.

## 3) Métricas

A fim de avaliar nossos modelos, devemos sempre utilizar algumas métricas de performance. No nosso contexto, queremos classificar as fotos em três diferentes classes, dessa forma decidimos usar a Precisão e Recall por label, juntamente com a Acurácia na avaliação. Segue abaixo as métricas:

- Pra cada uma das classes (c), será calculada a precisão e recall seguindo as fórmulas abaixo:

$$Precision(c) = \frac{TP(c)}{TP(c) + FP(c)}$$

$$Recall(c) = \frac{TP(c)}{TP(c) + FN(c)}$$

- E a acuŕacia também será considerada na avaliação:

$$Acc = \frac{TP + TN}{TP + TN + FP + FN}$$

Como não há nenhum grande desbalanceamento entre as classes, a acurácia será utilizada como a métrica mais importante. Se houver algum empate, consideraremos precisão e recall, respectivamente.

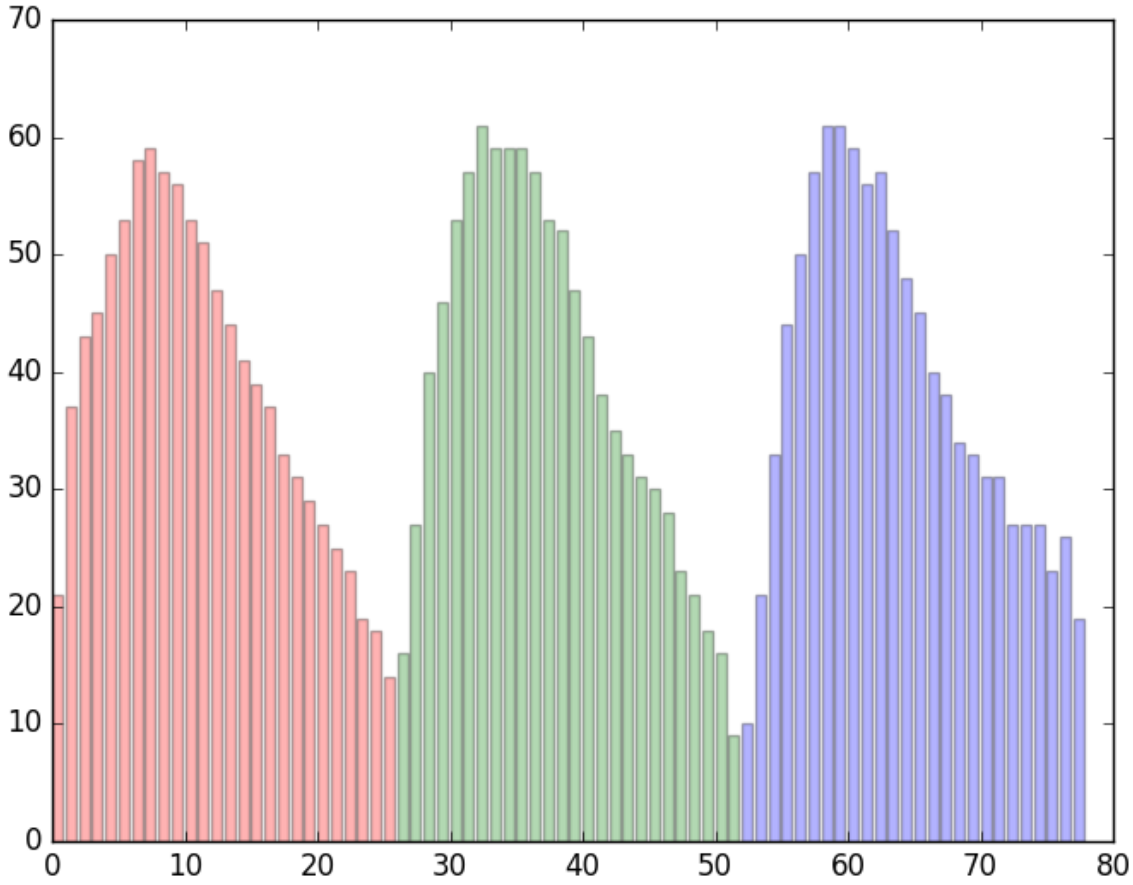
## 4) Modelo de Benchmark

O algoritmo conhecido como [KNN](https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm) foi escolhido como o modelo de benchmark. A ideia desse algoritmo é utilizar dos "K" vizinhos mais próximos de um determinado dado a fim de classificá-lo.

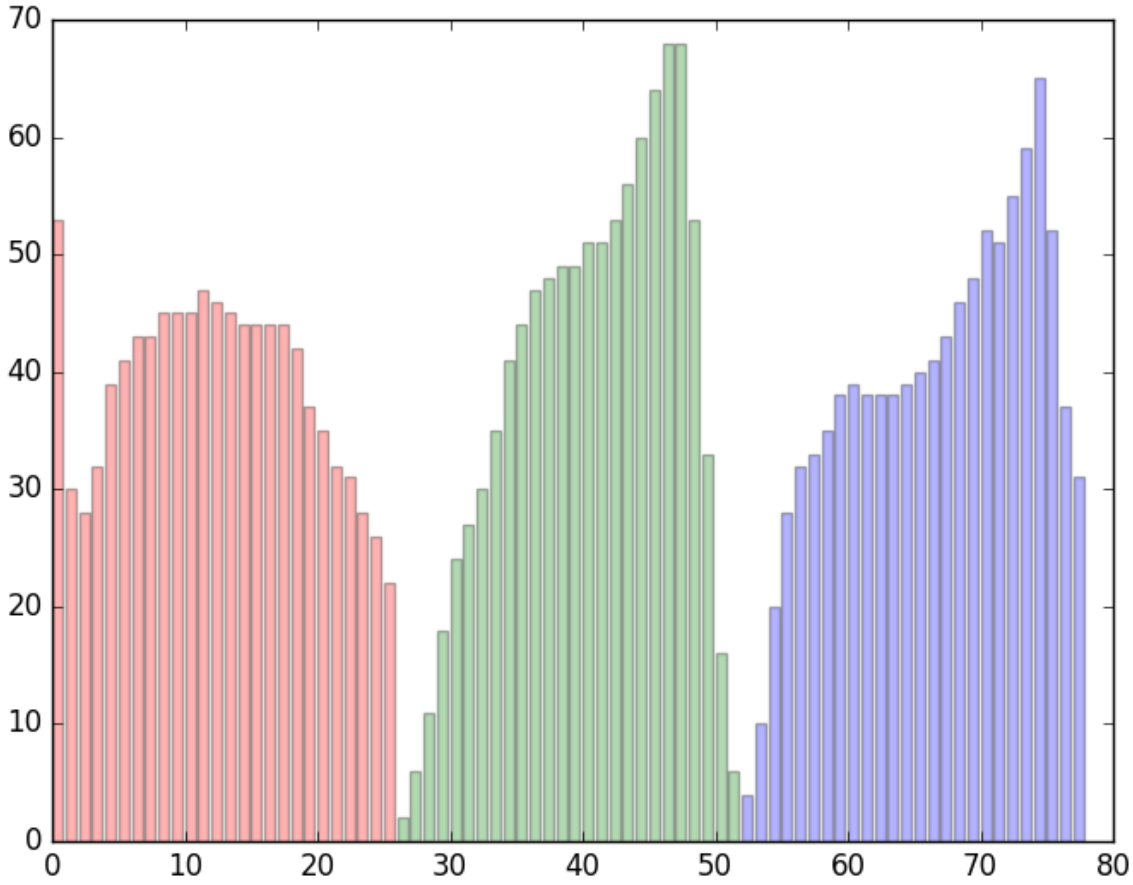
### 4.1) Pré-processamento

Antes de aplicar KNN nos dados, algum pré-processamento foi necessário, a fim de transformar as imagens em um formato que possa ser aceito pelo algoritmo. Contudo, é bom lembrar que estamos tentando criar um modelo que possa identificar três diferentes padrões de imagens, ou seja, precisamos processar elas, de forma que realce as diferenças de caracteristicas, o que por sua vez, facilita o aprendizado do modelo. Sabemos que uma imagem pode ser representada como uma matrix de pixels que variam de 0 até 255. Todas as nossas imagens são coloridas e seguem o padrão RGB (Red, Green e Blue), ou seja, cada imagem é composta por, na verdade, três matrizes de pixels, no qual cada uma remete à uma das três cores. Com isso em mente, uma boa estratégia é usar um histograma de cores de cada imagem como features para o modelo, visto que cada um dos ambientes apresenta uma distribuição particular.

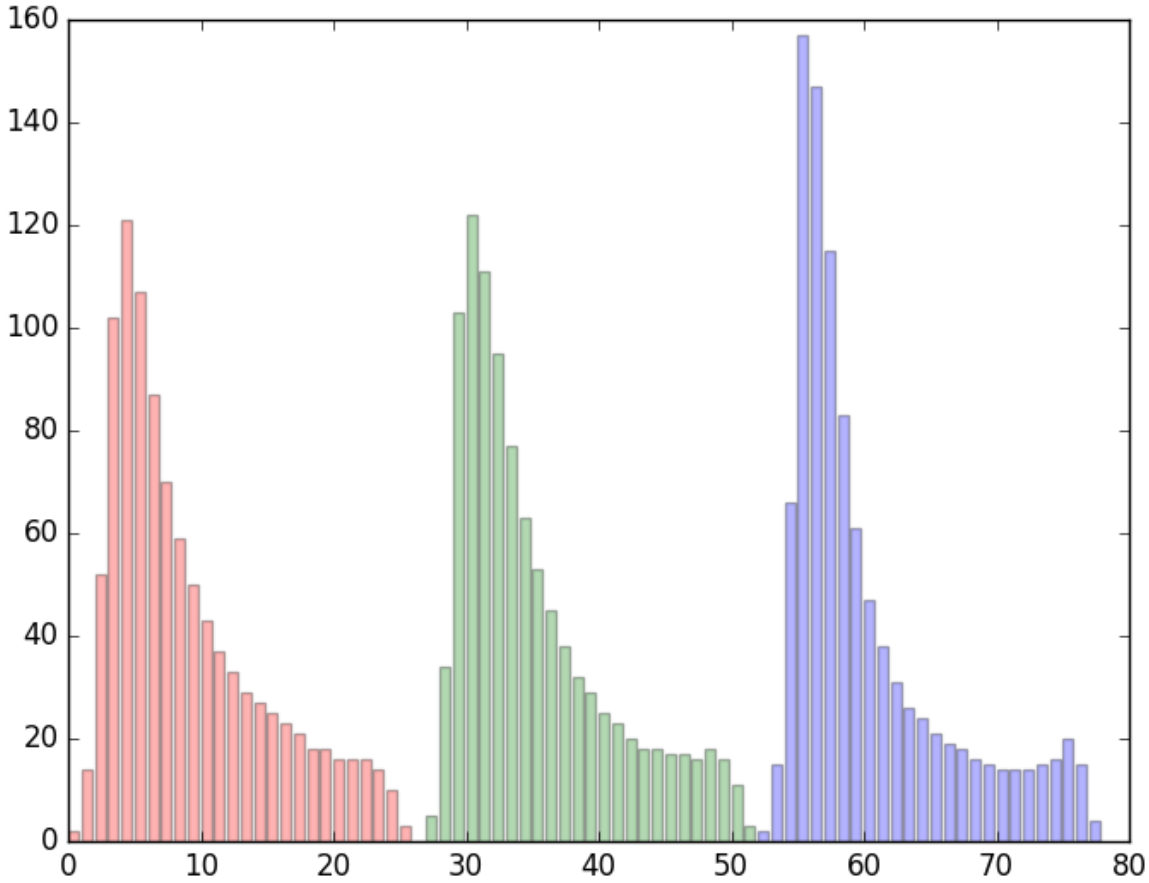
**Observação**: O histograma original de cada imagem se tornou dificil de processar utilizando o laptop que se encontra à disposição, pois o mesmo acabava criando um demasiado espaço de features. Por consequência, foi necessário diminuir o tamanho do histograma à partir do aumento dos *bins* dos mesmos. As imagens abaixo representam a distribuição de cores de cada imagem após tal processamento. Note que cada cor representa uma das cores RGB. Além disso, todas as imagens foram redimensionadas para tamanho **32x32** logo antes da criação dos histogramas.




**Histograma de cores para imagens urbanas**




**Histograma de cores para imagens de praias**




## Histograma de cores para imagens de florestas

Observando os histogramas acima, vemos claramente que as distribuições são, realmente, distinguíveis entre si. É perceptível que nas imagens relacionadas com florestas e cidades, a distribuição das cores ocorre de forma mais similar entre elas, ao passo que no contexto de praias, não é tão similar, visto que a distribuição do vermelho destoa bastante do verde e azul.

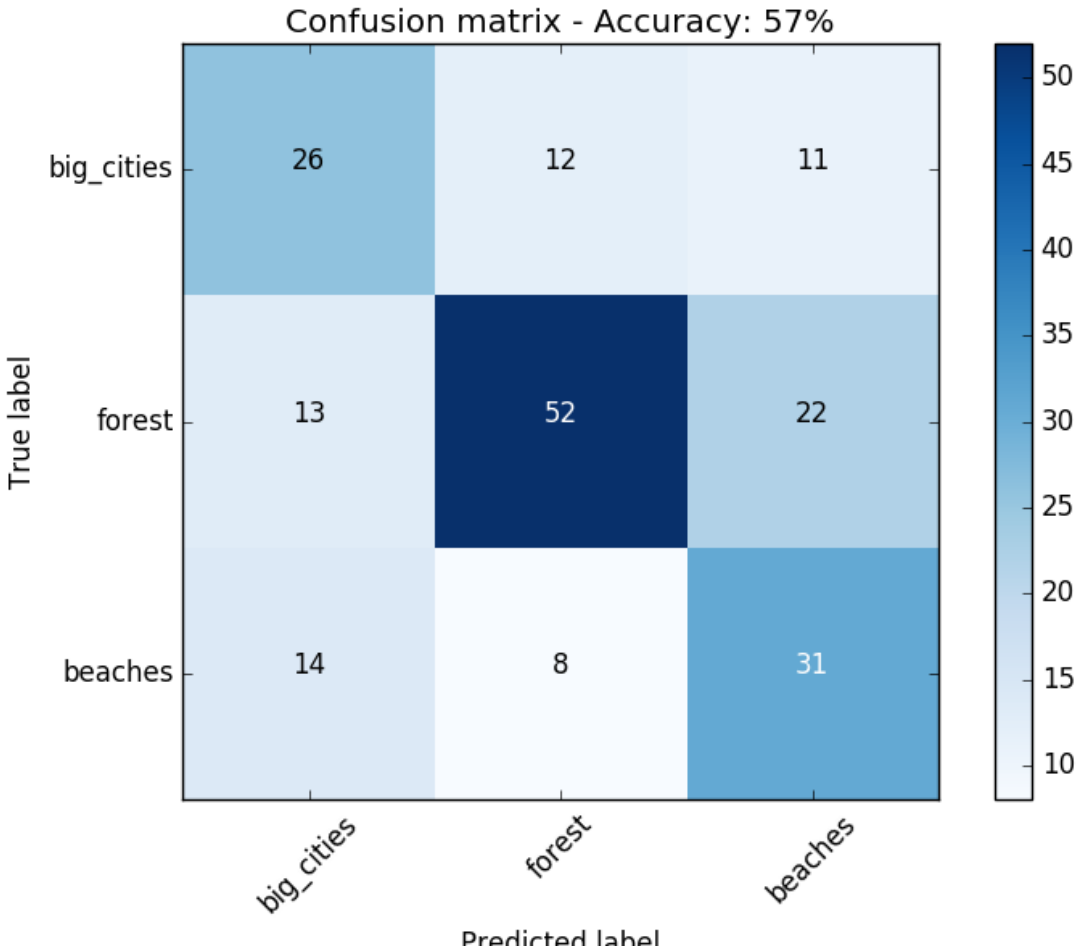
**4.2) Treinando o modelo de benchmark: KNN**

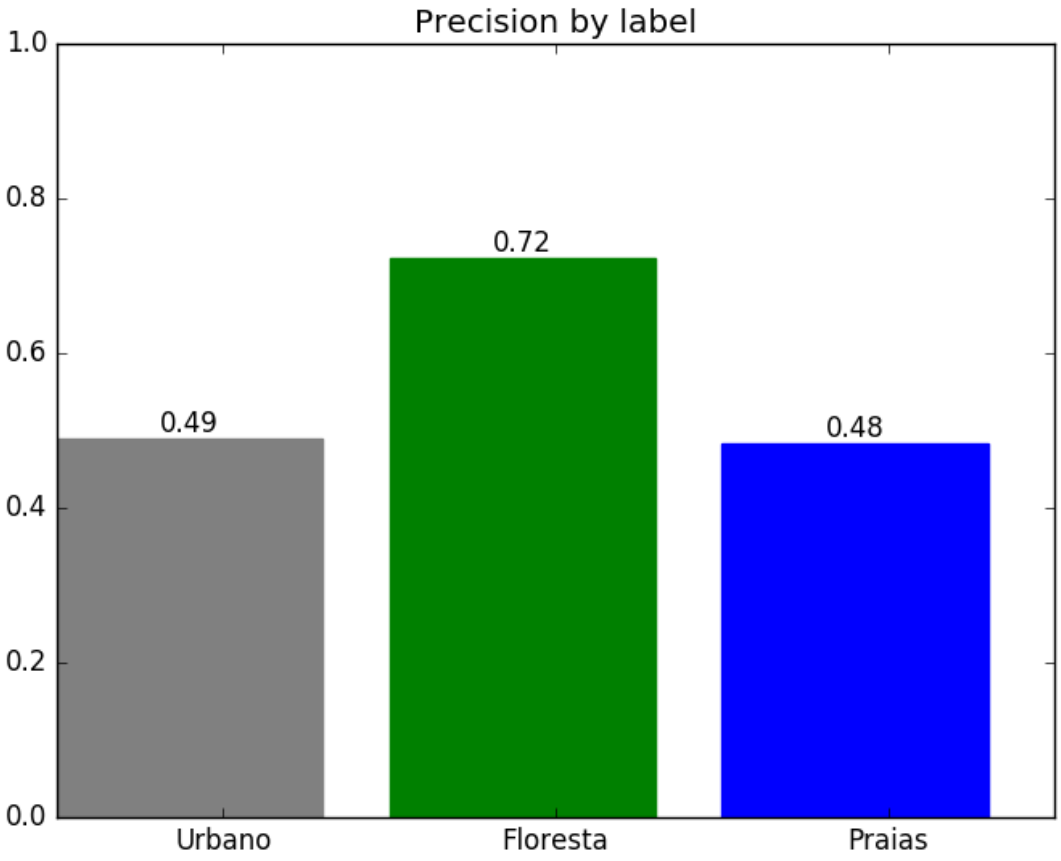
O treinamento do KNN usa como entrada a distribuição de cores de cada imagem. Sendo que, foi aplicado um algoritmo conhecido como _grid search_, a fim de encontrar a melhor quantidade de vizinhos a ser considerada quando classificar uma imagem. Além disso, 80% das imagens foram usadas como dados de treino e o resto como teste. O parâmetro *random_split*, também, foi utilizado para caso seja necessário replicar a separação de treino e teste, e, com isso, facilitar a comparação com outros modelos. Os resultados das métricas podem ser visualizados abaixo:

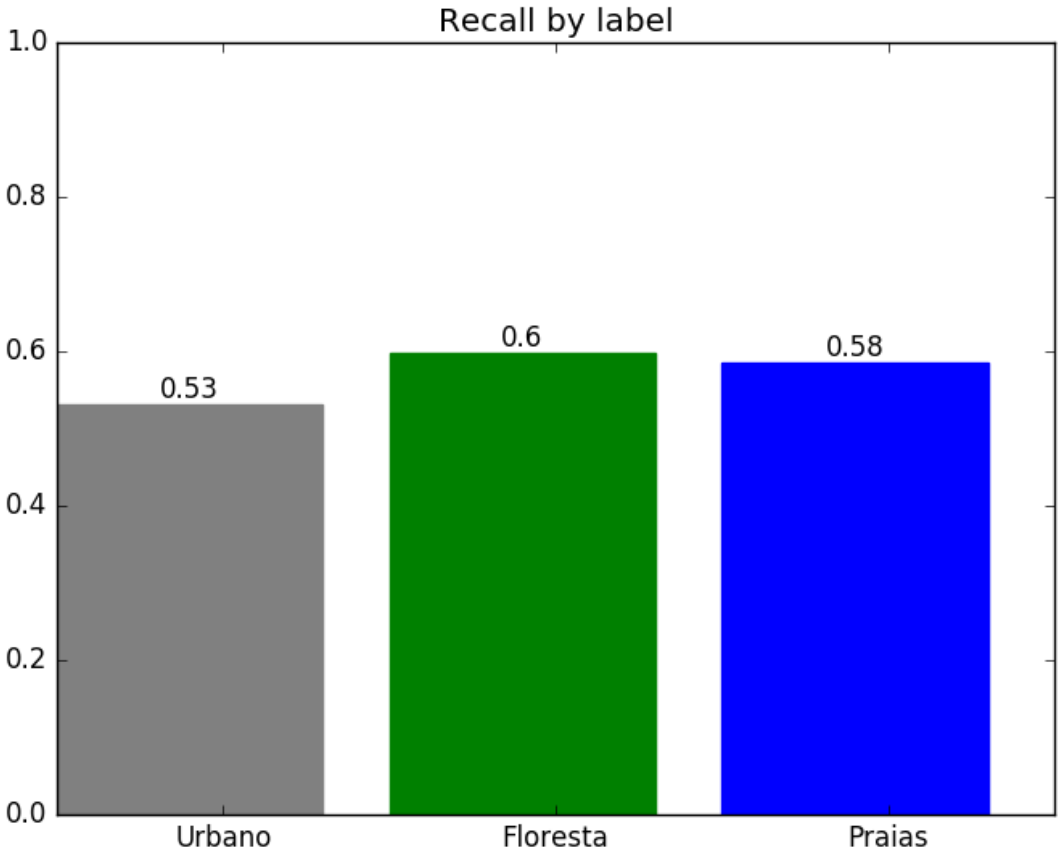
- Quantidade de imagens usadas no treino: **756**
- Quantidade de imagens usadas no teste: **189**
- Configuração do melhor modelo

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=12, p=2,
           weights='uniform')
```

- Resultados da aplicação do modelo nos dados de teste pode ser visto. Tais resultados serão usados como parâmetro para indicar se os outros modelos criados obtiveram ou não sucesso.




Precision by label
1.0
0.8
0.6
0.4
0.2
0.0
0.49
0.72
0.48
Urbano
Floresta
Praias


Recall by label
1.0
0.8
0.6
0.4
0.2
0.0
0.53
0.6
0.58
Urbano
Floresta
Praias

## 5) Deep Learning

*Deep learning* é um conjunto de técnicas e algoritmos que envolve o uso de redes neurais profundas, ou seja, com muitas camadas. Esses tipos de redes neurais ganharam bastante atenção após os seus ótimos desempenhos nos campos de reconhecimento de imagem e voz. Pensando nisso, foi desenvolvido e avaliado duas arquiteturas de modelos _deep learning_ nesse projeto. *Keras* foi o framework utilizado na construção de tais modelos.

### 5.1) Pré-processamento

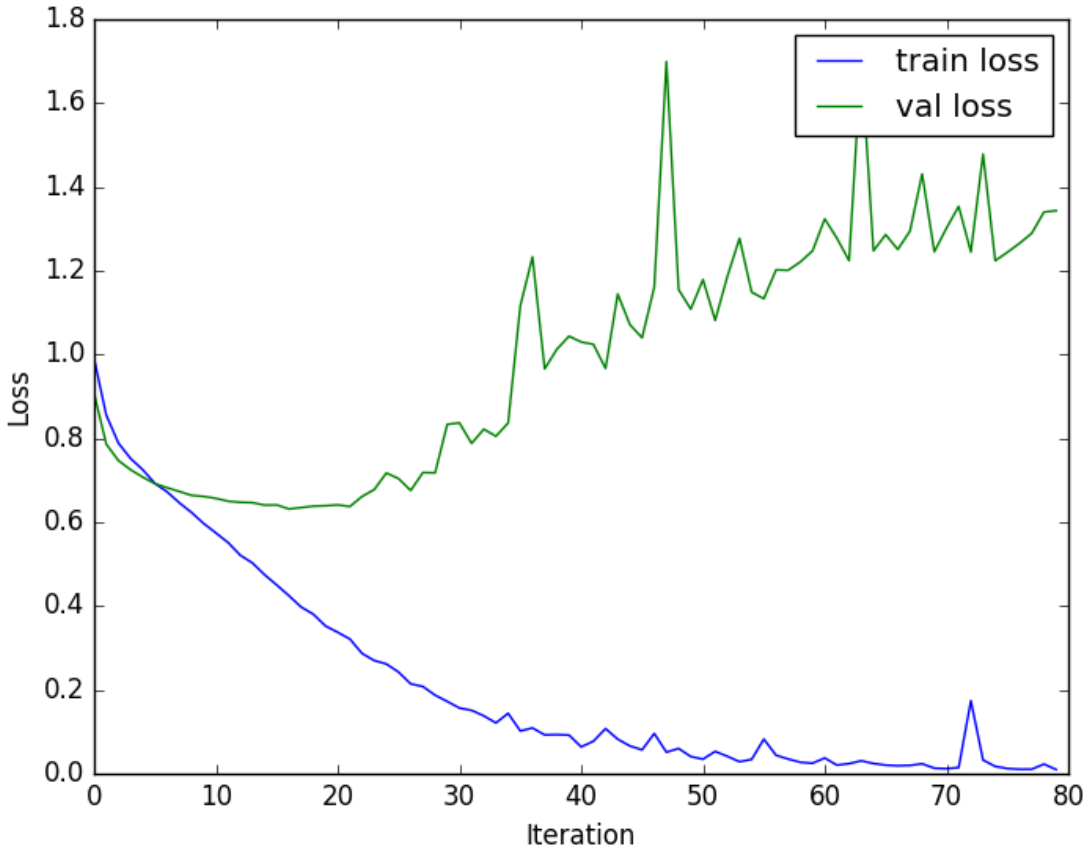
Todas as imagens foram redimensionadas para o tamanho de **32x32** e os pixels foram colocados no intervalo de zero à um pela divisão dos mesmo por 255.

### 5.2) Primeira arquitetura

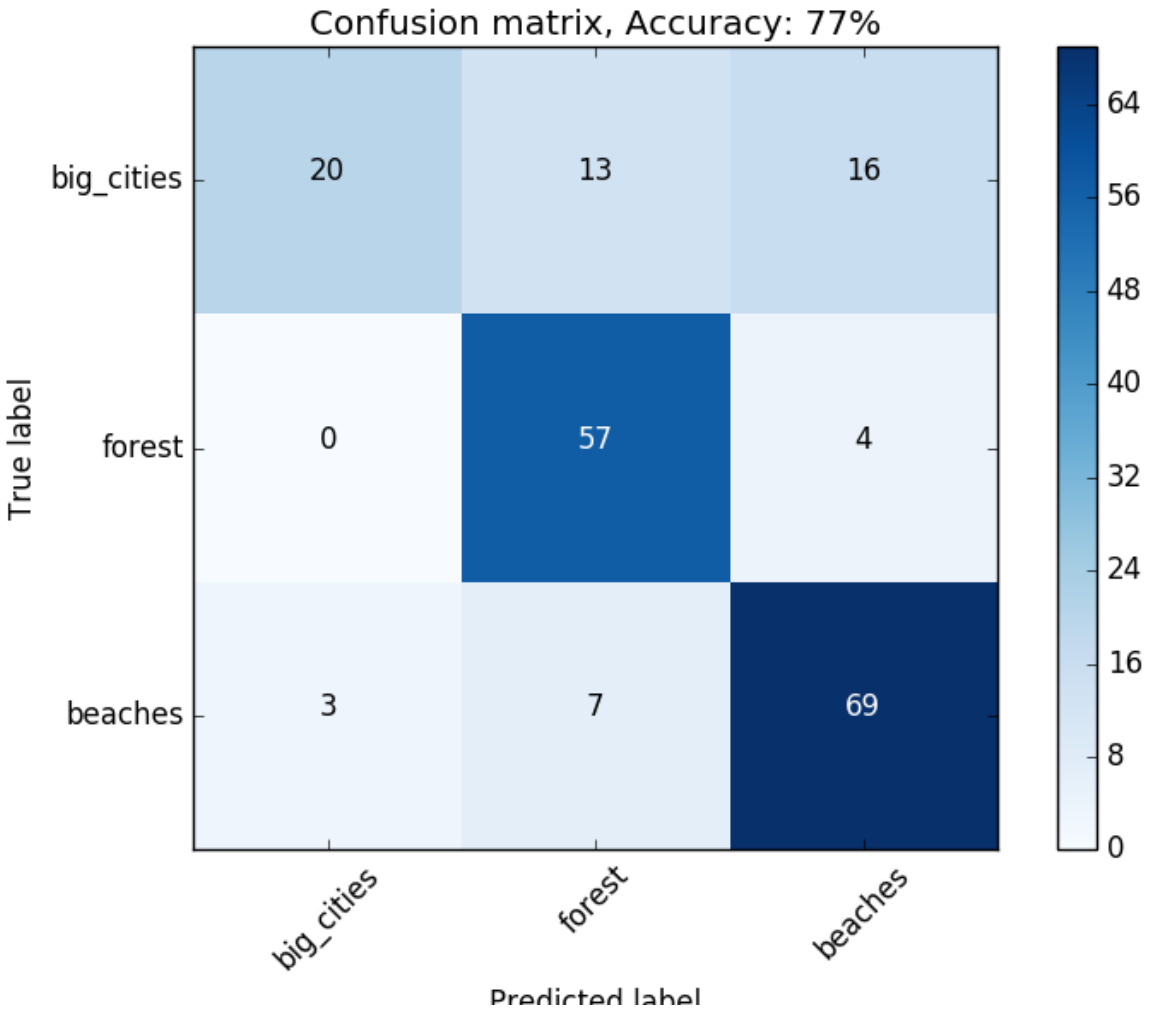
- A primeira arquitetura construída e avaliada contém três camadas *Denses* com 500, 300 e 100 neurônios cada, seguida por uma camada *Flatten* com 102400 neurônios e, por fim, a camada de softmax com 3 neurônios. Ademais, uma camada de *Dropout* foi adicionada entre cada camada *Dense* a fim de evitar *overfitting*. Tal arquitetura pode ser melhor sumarizada na imagem abaixo.

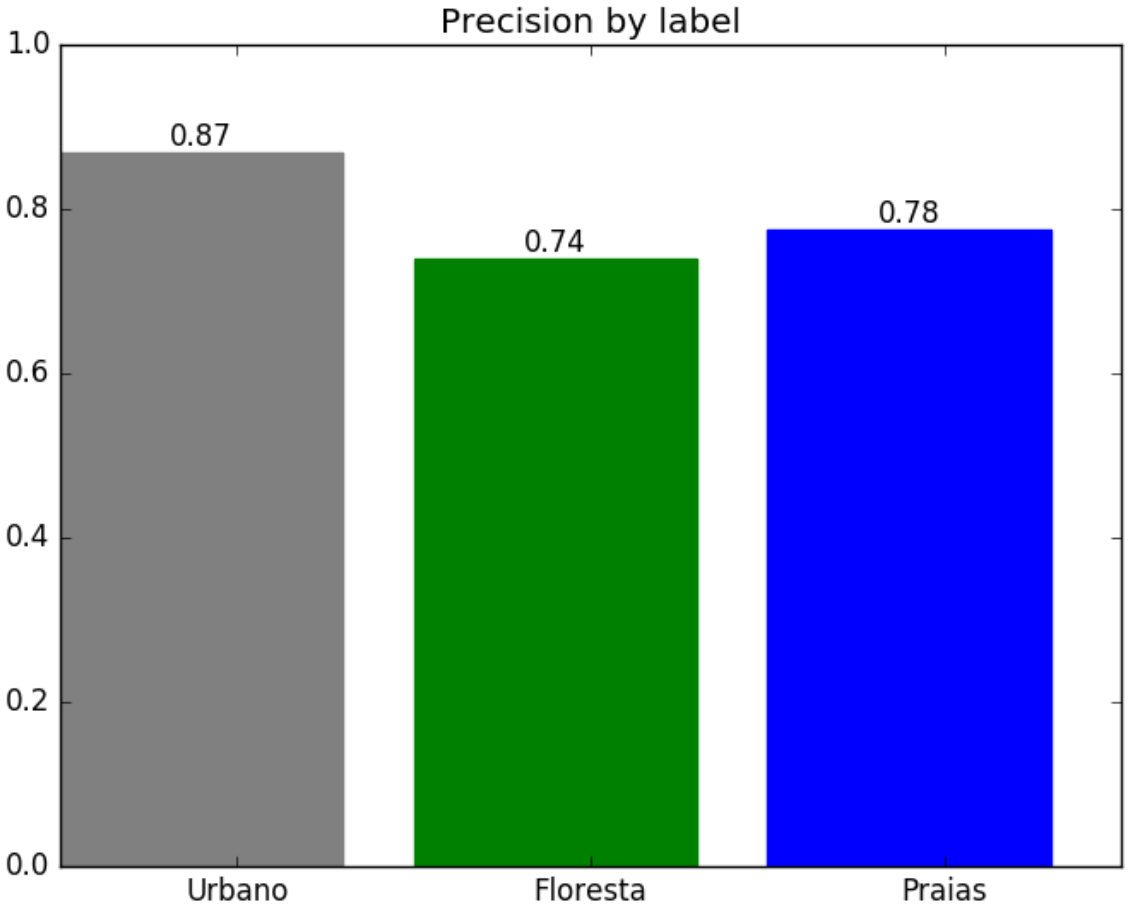
| Layer (type) | Output Shape | Param # |
|---|---|---|
| dense_9 (Dense) | (None, 32, 32, 500) | 2000 |
| dropout_1 (Dropout) | (None, 32, 32, 500) | 0 |
| dense_10 (Dense) | (None, 32, 32, 300) | 150300 |
| dropout_2 (Dropout) | (None, 32, 32, 300) | 0 |
| dense_11 (Dense) | (None, 32, 32, 100) | 30100 |
| dropout_3 (Dropout) | (None, 32, 32, 100) | 0 |
| flatten_3 (Flatten) | (None, 102400) | 0 |
| dense_12 (Dense) | (None, 3) | 307203 |

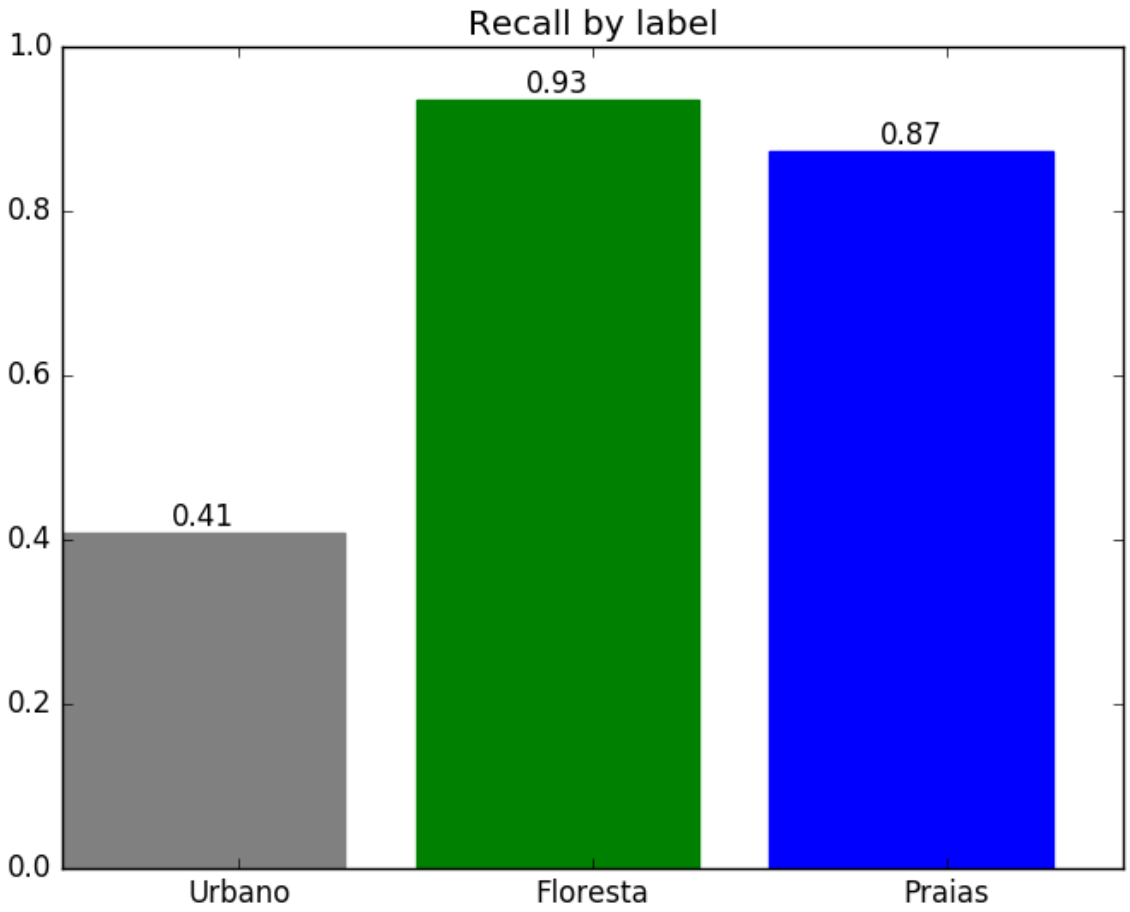
- O primeiro experimento foi executado com 80 épocas e a curva do erro no teste e treino pode ser vista abaixo.




- Observando o gráfico acima, notamos que não precisamos usar tantas épocas, visto que o erro do teste para de descer por volta da época vinte e quatro. Por isso, resolvemos rodar novamente o experimento com vinte e quatro épocas apenas. Os resultados da aplicação do modelo nos dados de teste pode ser visto abaixo.


Confusion matrix, Accuracy: 77%
big_cities
forest
beaches
True label
20
13
16
0
57
4
3
7
69
64
56
48
40
32
24
16
8
0
big_cities
forest
beaches
Predicted label


Precision by label
1.0
0.8
0.6
0.4
0.2
0.0
0.87
0.74
0.78
Urbano
Floresta
Praias




Os resultados acima já indicam uma boa evolução em relação ao modelo de benchmark. Lembrando, que a divisão de dados de treino e teste foi mantida, pois dessa forma, conseguimos fazer uma comparação justa entre os modelos.

#### 5.3) Segunda arquitetura

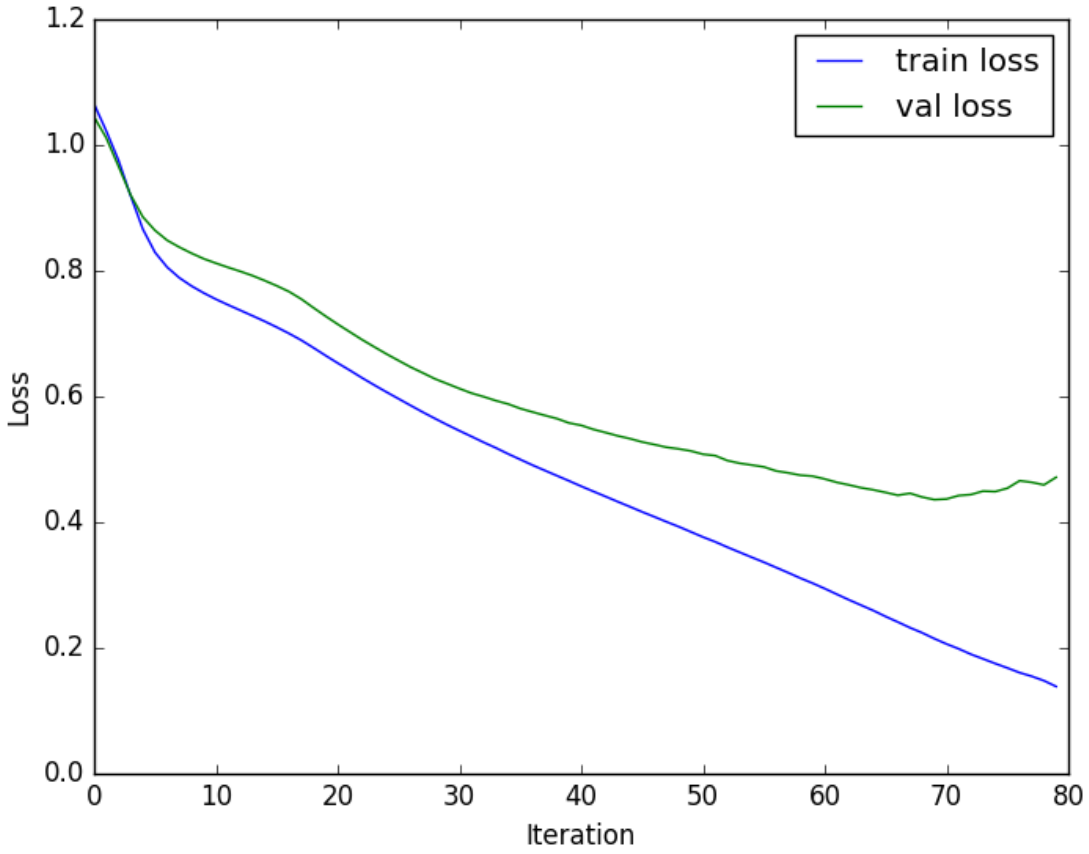
A segunda arquitetura foi contruída em cima dos conceitos de *Convolutional Neural Networks* e *Max Pooling*. Abaixo segue uma explicação sobre tais conceitos e o porquê eles foram escolhidos.

- **Camadas convolucionais**: Tais camadas são muito úteis quando queremos discriminar melhor os padrões que estamos tentando ensinar à nossa rede neural. Elas funcionam por meio da convolução/deslizamento de um filtro sobre a imagem original. Dessa forma, esse filtro é capaz de identificar diferentes padrões em diferentes partes da imagem. Esse tipo de camada, diferentemente das camadas *Denses*, é localmente conectada, ou seja, seus neurônios são conectados à um subconjunto de neurônios da camada anterior, o que diminui bastante a complexidade da rede.
- **Camadas do tipo Pooling**: Tais camadas são, geralmente, aplicadas, logo após, uma camada de convolução. Elas funcionam de forma similar as camadas convolucionais, no qual, em ambas fazemos o deslizamento de um filtro. Nesse projeto, nós focamos em camadas do tipo *Max*, onde as mesmas selecionam o pixel de maior valor dentro da janela de filtro para que esse seja considerado o valor do nó. A vantagem dessa técnica é a redução de dimensionalidade causada pelo filtro que, consequentemente, gera uma redução do custo computacional de construção o modelo, visto que diminui a quantidade de parâmetros. Além disso, conseguimos aumentar as chances de evitar um possível *overfitting*, visto que temos menos parâmetros.

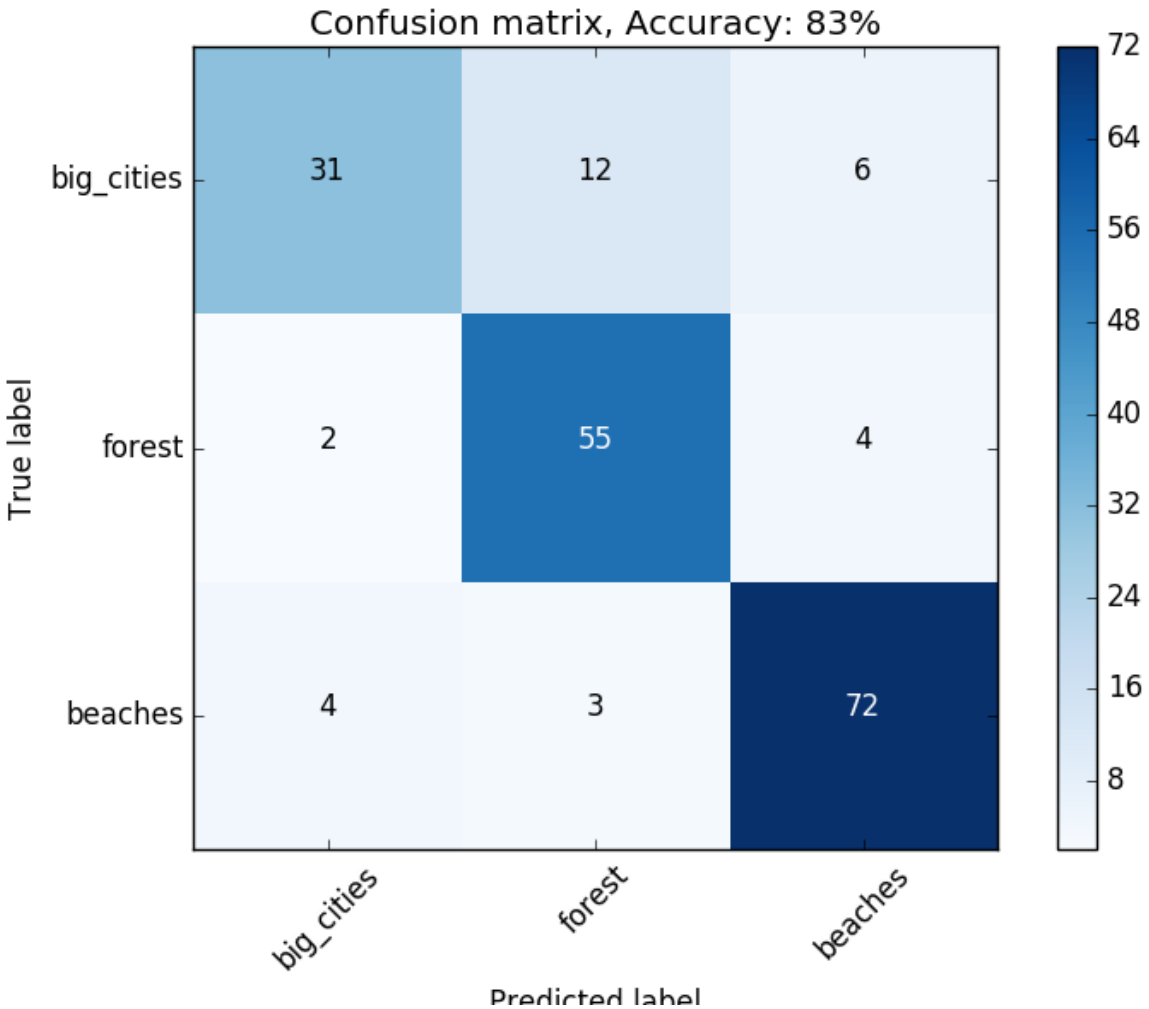
Redes neurais convolucionais vem apresentando excelentes resultados no campo de classificação de imagens, e por isso, decidimos aplicar tal técnica nesse projeto. A arquitetura sumarizada da rede pode ser vista abaixo:

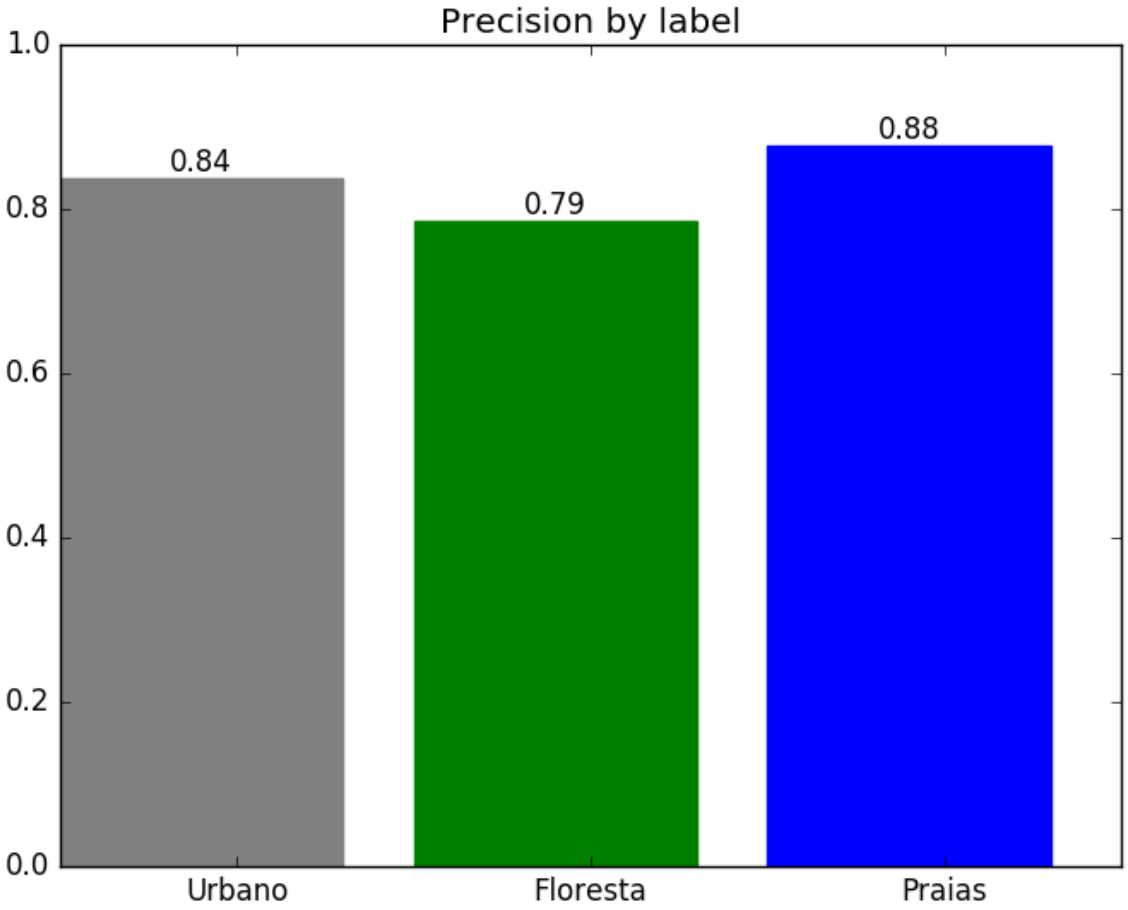
| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d_1 (Conv2D) | (None, 32, 32, 16) | 208 |
| max_pooling2d_1 (MaxPooling2 | (None, 16, 16, 16) | 0 |
| conv2d_2 (Conv2D) | (None, 16, 16, 32) | 2080 |
| max_pooling2d_2 (MaxPooling2 | (None, 8, 8, 32) | 0 |
| conv2d_3 (Conv2D) | (None, 8, 8, 64) | 8256 |
| max_pooling2d_3 (MaxPooling2 | (None, 4, 4, 64) | 0 |
| flatten_1 (Flatten) | (None, 1024) | 0 |
| dense_1 (Dense) | (None, 500) | 512500 |
| dense_2 (Dense) | (None, 3) | 1503 |

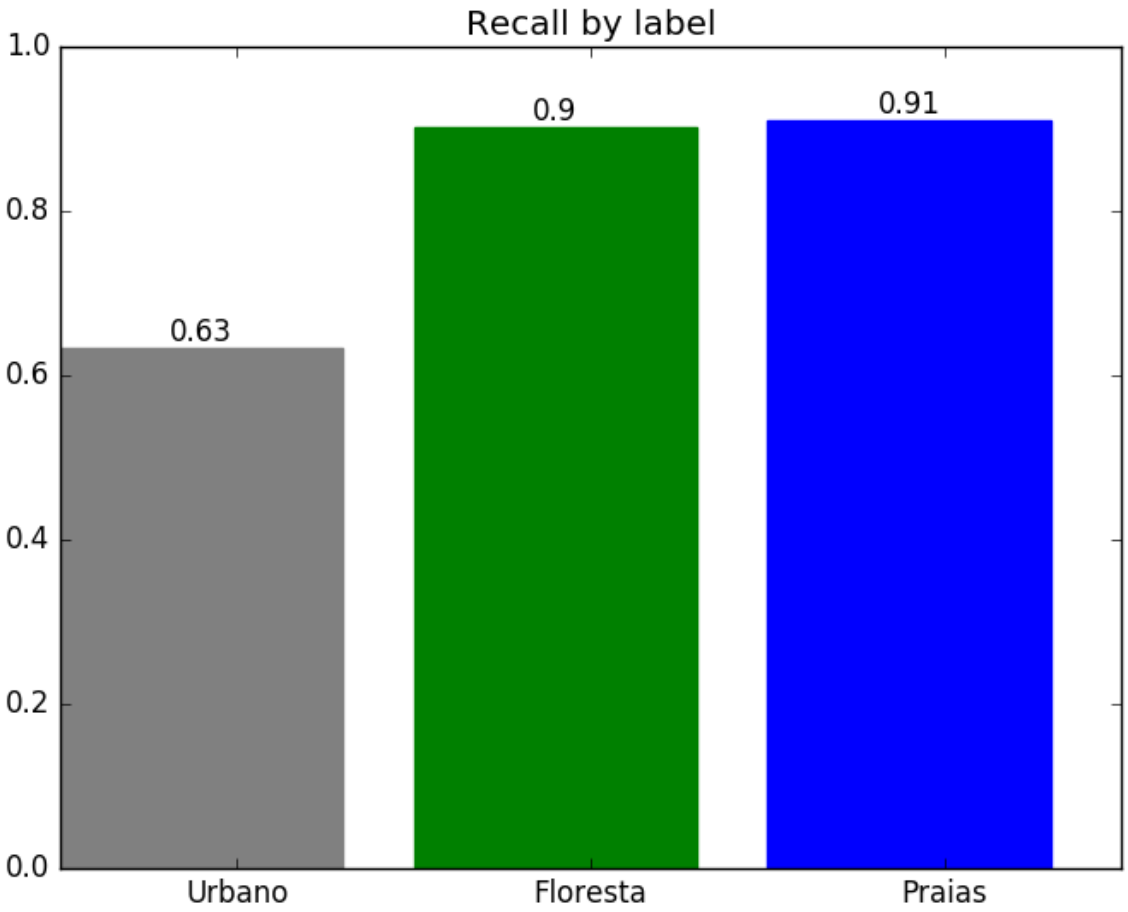
- Gráfico da curva do erro.




- O modelo escolhido foi treinado com 80 épocas, sendo que observando a curva do erro no gráfico acima, não sentimos necessidade de diminuir a quantidade de épocas. Os resultados da aplicação de tal modelo nos dados de teste pode ser visto logo abaixo.


Confusion matrix, Accuracy: 83%
big_cities
forest
beaches
True label
31
12
6
2
55
4
4
3
72
72
64
56
48
40
32
24
16
8
big_cities
forest
beaches
Predicted label


Precision by label
1.0
0.8
0.6
0.4
0.2
0.0
0.84
0.79
0.88
Urbano
Floresta
Praias


Recall by label
1.0
0.8
0.6
0.4
0.2
0.0
0.63
0.9
0.91
Urbano
Floresta
Praias

## 6) Sumarização dos resultados

Os resultados das métricas foram sumarizados na tabela abaixo:

| Métricas | Benchmark | DL-Camadas denses | DL-Camadas convolucionais |
|---|---|---|---|
| Precisão | 56% | 79% | 83% |
| Revocação | 57% | 73% | 81% |
| Acurácia | 57% | 77% | 83% |

*Obs: O valor das métricas de revocação e precisão foram calculadas a partir da média da precisão e revocação de cada classe.

Fica evidente que os resultados da segunda arquitetura foram melhores do que da primeira, e substancialmente, melhores do que o nosso modelo de *benchmark*, visto que ela se mostrou superior não somente em relação a acurácia, mas em precisão e revocação também. Abaixo, segue uma aplicação do melhor modelo em algumas fotos com sua respectiva predição e probabilidade:

**Prediction:** Urban | **Probability** : 59%

**Prediction** : Beache | **Probability** : 99%

**Prediction** : Forest | **Probability** : 100%

## 7) Conclusão

Nesse projeto, demonstramos ser possível utilizar uma arquitetura de *deep learning* para classificar imagens de três ambientes diferentes. Verificamos, também, que o uso de técnicas de convolução melhorou a performance do modelo, sendo tal arquitetura escolhida como a melhor. Ainda assim, acreditamos que o modelo ainda possa ser evoluido. Aumentar a base de imagens, que atualmente apresenta 945 fotos, seria uma boa alternativa, haja visto que técnicas de *deep learning* possuem melhores resultados quando aplicadas em grandes bases. Isso não foi aplicado nesse projeto devido à pouca capacidade computacional que tínhamos disponível.

Além disso, vale ressaltar que todo o fluxo de trabalho necessário para construção de um modelo de aprendizagem de máquina foi seguido. Começando desde o desenvolvimento da base de dados, passando pelo treinamento dos modelos e chegando na escolha do melhor. É interessante deixar grifado que um tempo significante foi despendido na obtenção da base de treino de imagens, no qual a mesma foi construída a partir de conhecimentos de programação e técnicas de *web crawler*. O que corrobora a ideia de que a área de ciência de dados e aprendizagem de máquina em geral é bastante multidisciplinar por, muitas vezes, exigir que o profissional da área apresente conhecimentos estatisticos, matemáticos e de programção.

## 8) Referências

Keras Documentação (https://keras.io/)