Landmark Recognition

Grupo Neurobook

Eduardo Armelin
Ivan Bravin
Marcio Pinheiro Neris
Tiago Deliberali Santos

1. Objetivo

O objetivo do nosso trabalho é a identificação de pontos de referência em fotografias, utilizando um dataset desenvolvido pelo time do Google AI para treinamento do modelo. Esse desafio foi hospedado no Kaggle com o nome de Landmark Recognition.

A ideia desse desafio era oferecer para à comunidade um conjunto extremamente amplo de imagens anotadas, com uma grande variedade de escalas, condições e lugares, permitindo a pesquisadores ao redor do mundo avançarem com pesquisas nessa área.

2. Dados

O dataset oferecido pela competição é composto por 500 arquivos compactados de aproximadamente 1GB cada um. No total, temos 4.132.914 imagens distribuídas em 203.094 classes diferentes.

Abaixo um exemplo das imagens encontradas no dataset. Podemos notar a grande variedade de temas, condições e falta de padrão das fotos. Aqui, elas estão sendo exibidas com 224x224, formato usado por alguns dos nossos modelos.

Simplificação do problema

Para nosso problema, alinhamos com o Zanoni e Rafael que trabalhamos num conjunto de 100 classes e, dentro desse grupo, escolhemos um subconjunto de 20 classes. A seleção das 100 categorias foi feita pensando em maximizar o número de imagens por categoria, tentando reduzir um pouco o desbalanceamento entre classes. Para isso, ordenamos as classes por número de imagens e ignoramos as 49 primeiras. Assim, ficamos com classes que

possuíam entre 595 e 975 imagens. Por fim, fizemos uma seleção aleatória de 20 classes e separamos os grupos de treino e validação. Esse processo pode ser observado no arquivo **analise.py**.

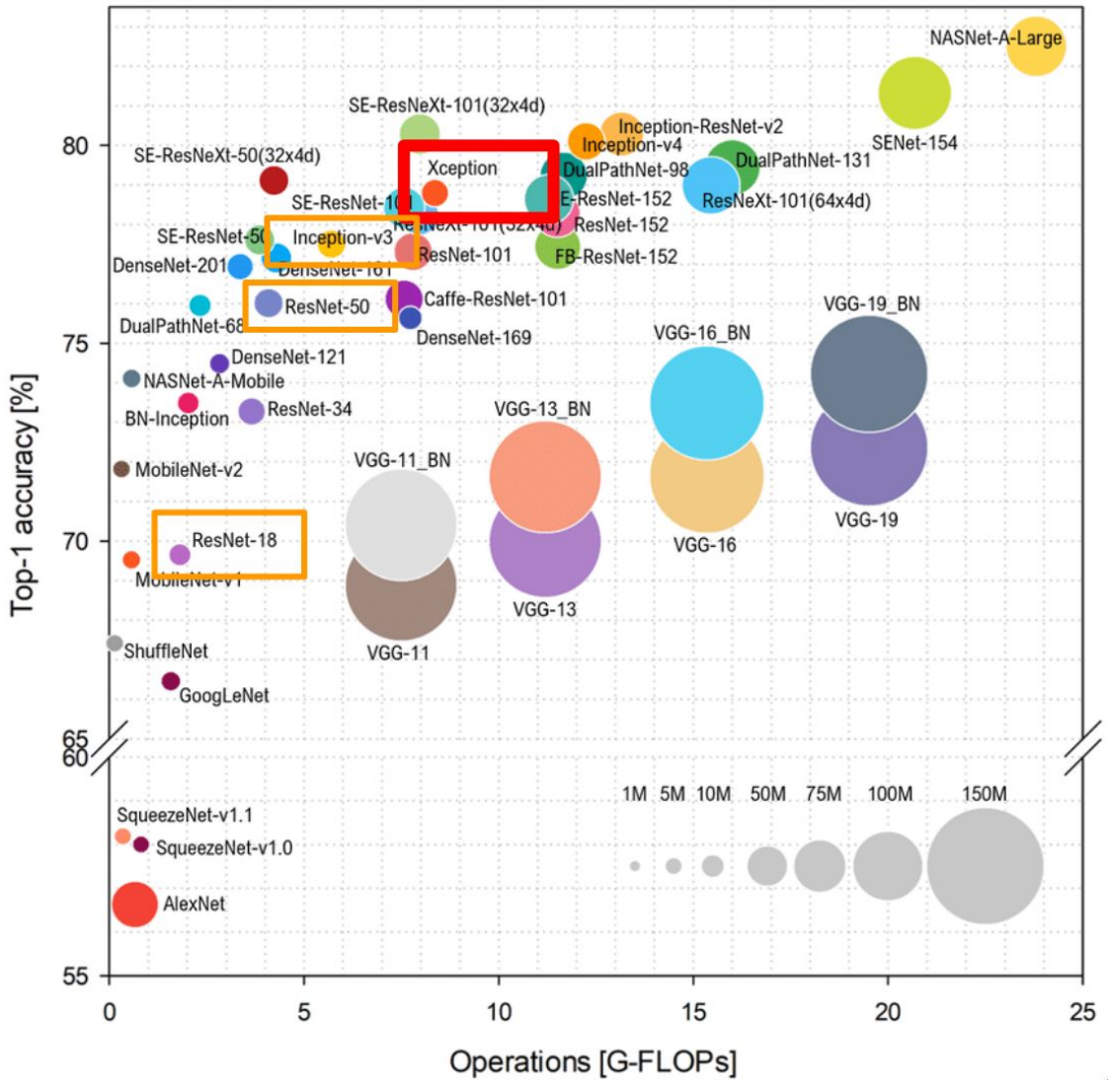
Para baixar os arquivos compactados para seleção das imagens que usamos, foi necessário subir uma máquina no Azure e executar o script fornecido pelo monitor Rafael, com uma pequena modificação para excluir os arquivos após a descompactação. Após termos as 4M de imagens descompactadas, utilizamos os csv's gerados no passo anterior para criar as pastas com as imagens de treino e teste para cada conjunto de classes. Com isso, terminamos o processo com 76.316 referentes às 100 classes e 15.636 imagens referentes às 20 classes. Esse processo foi feito utilizando os scripts **find_files.py** e **find_files_subset.py**.

Fatores limitantes

Todos os experimentos iniciais foram executados com o conjunto de 20 classes em máquinas hospedadas no AWS. Porém, quando tentamos repetir os experimentos com o conjunto de 100 classes, esbarramos em diversos erros de falta de memória, bugs no uso de múltiplos gpus no keras e erros de concorrência. Por mais que tentássemos utilizar máquinas cada vez maiores, não conseguimos fazer com que os notebooks funcionassem. Entendemos que faltou experiência no trato de máquinas com múltiplas placas de vídeo ou grandes montantes de memória RAM. Por isso, todas as análises do nosso trabalho se baseiam no conjunto de 20 classes.

3. Estratégias para solução do problema

Para a solução desse problema, optamos por utilizar a técnica de transfer learning, baseado em redes treinadas com dados da competição ImageNet. Foram consideradas redes que estivessem disponíveis dentro do Keras e que possuíam menor número de parâmetros, abaixo tempo de treinamento e boa acurácia.



Observando o diagrama acima, temos destacadas as 4 redes avaliadas e como elas se comportam em termos de custo de treino, acurácia e número de parâmetros. Outras redes foram descartados devido às limitações de

hardware e assim seus resultados não serão reportados, apesar da tentativa (exemplo NasNet-A-Large e InceptionResNet). Deste modo destacamos as seguintes redes:

- Inception-V3
- ResNet-18
- ResNet-50
- Xception

Para cada uma dessas redes, padronizamos a seguinte sequência de eventos:

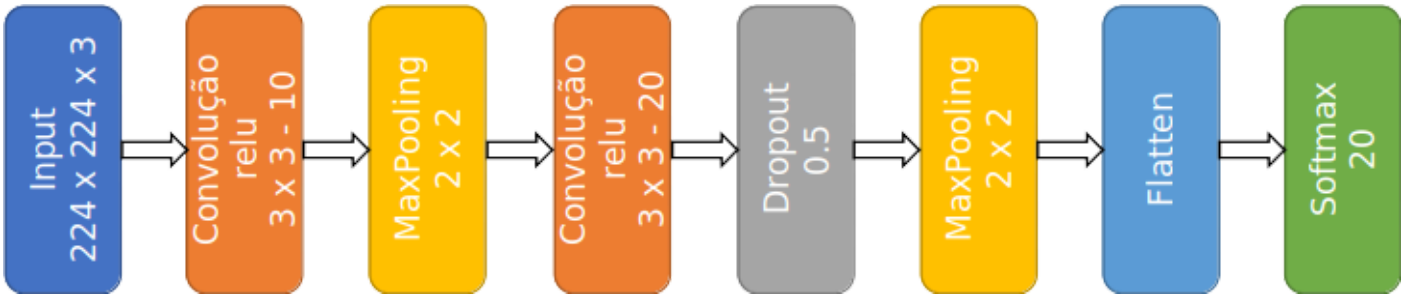
1. Encontrávamos a rede no Keras, com os pesos da ImageNet
2. Criamos uma instância sem a camada densa, com a opção do Keras de *include_top = False*
3. Congelamos as camadas deste modelo
4. Construímos uma rede densa de 3 camadas e associamos ao modelo congelado
5. Treinamos a rede para aquecimento dos pesos da rede densa
6. Descongelamos aa rede e treinamos ela novamente

A arquitetura da nossa rede densa adicionada aos modelos do Keras seguiu a seguinte arquitetura:

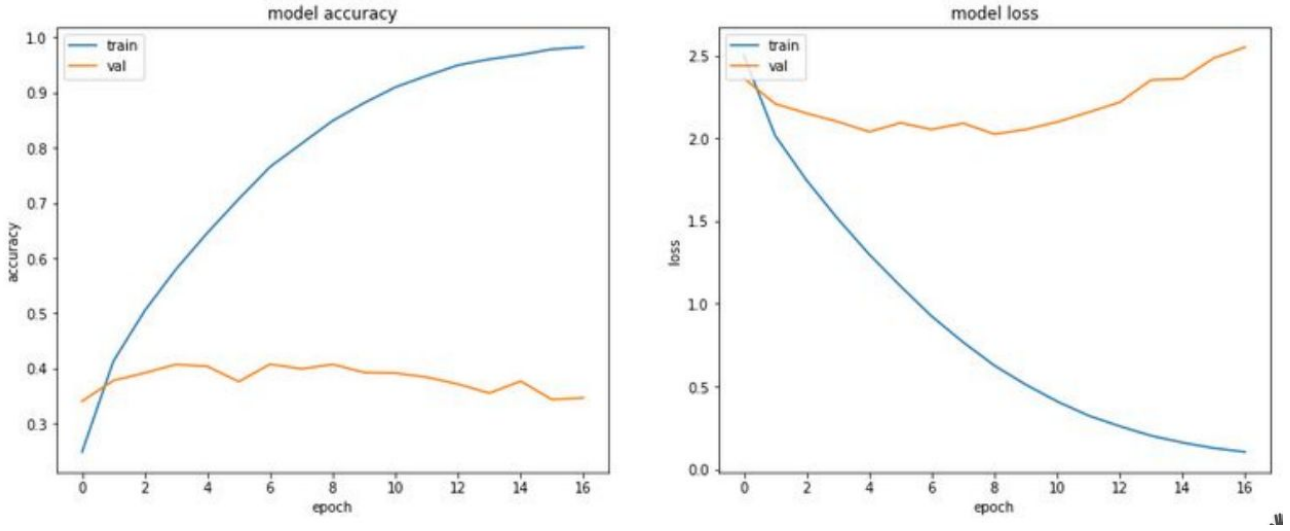
1. Global average pooling
2. Camada densa com 512 neurônios e função de ativação ReLU
3. Camada densa com 20 neurônios e Softmax

4. Treinando e avaliando os modelos na base de validação

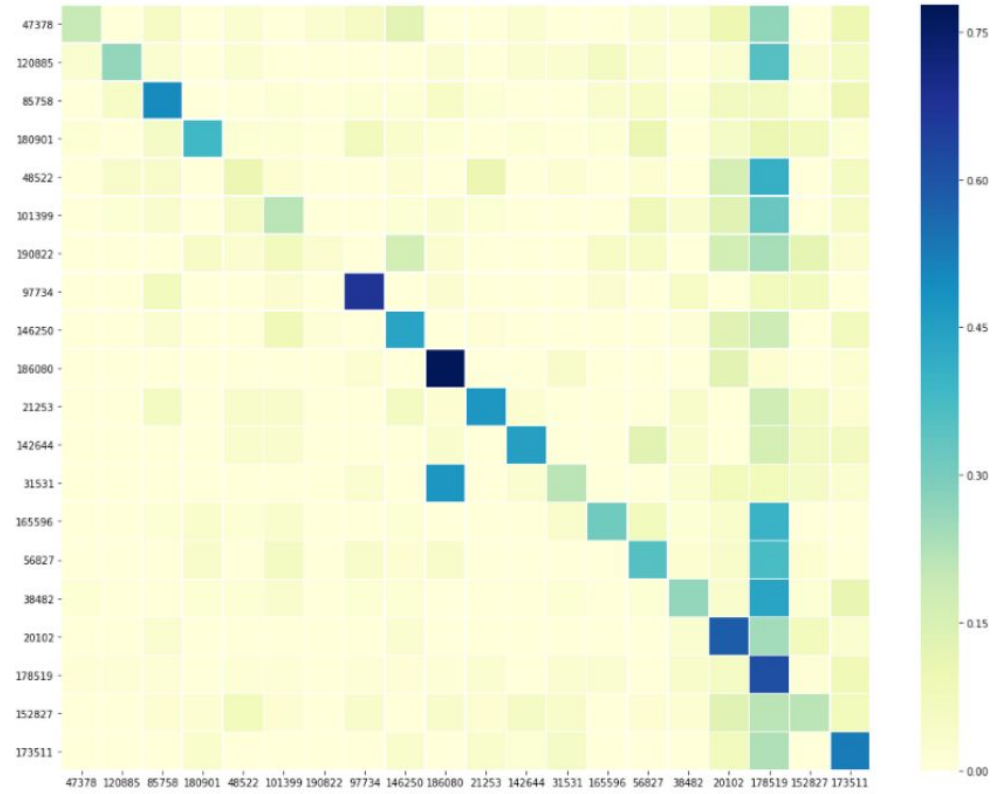
Para comparação dos resultados, construímos uma baseline baseada nos exemplos vistos em aula. Ela é uma rede simples, composta por duas camadas convolucionais e uma camada densa usando softmax para previsão das 20 classes.



Essa rede é constituída de cerca de 1M de parâmetros e apresentou os seguintes resultados:



Observando a matriz de confusão, vemos uma diagonal marcada, mas ainda bastante dispersa, com muitos erros de previsão. Olhando para as classes em que o modelo mais acertou a classificação, o baseline conseguiu um acerto de 78%, mas a performance se degrada rapidamente.



Top 10

classes	acurácia
186080	0,7778
97734	0,6744
178519	0,6164
20102	0,5854
173511	0,5263
85758	0,5
21253	0,4706
142644	0,4516
146250	0,4394
180901	0,3860

Acurácia de treino:

83.26%

Acurácia de validação:

37.34%

Resumo dos resultados dos modelos

Executamos uma série de experimentos documentados nos notebooks anexados ao trabalho. Aqui, apresentamos uma tabela com o resumo dos resultados que encontramos:

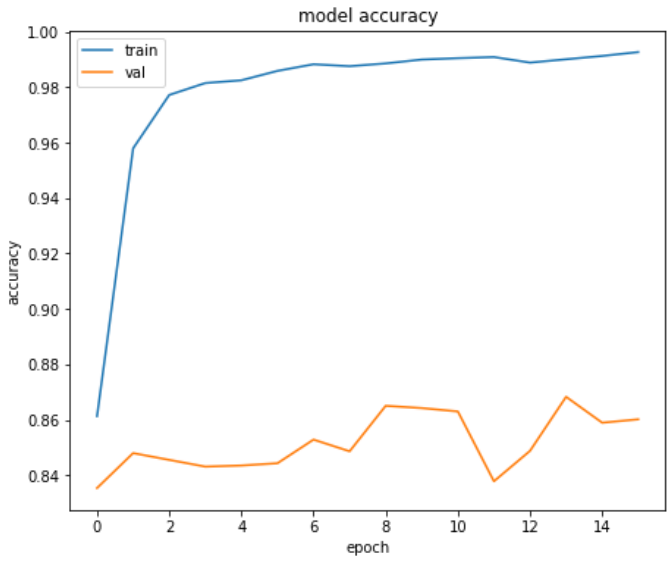
CNN	Acurácia		Parâmetros		
	Treino	Validação	Treináveis	Não Treináveis	Total
Homemade(Baseline)	83,26%	37,34%	1M	0	1M
ResNet18	93,26%	72,29%	12M	8k	12M
ResNet50	98,27%	79,50%	24M	53k	24M
Xception	99,54%	85,95%	21M	50k	21M
Inception V3	93,36%	76,64%	22M	34k	22M

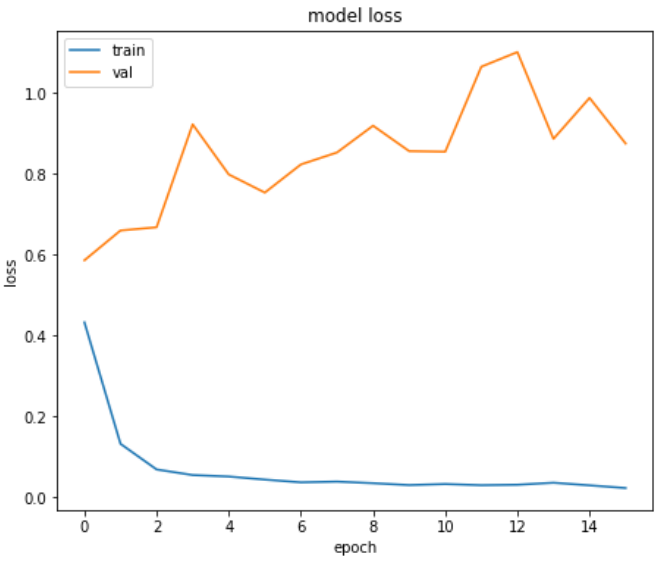
É interessante observar como esses resultados conversam com o diagrama de acurácia por esforço de treinamento apresentado. Lá, podíamos ver que a ResNet-18 era a rede com menor acurácia, enquanto a Xception era a rede com maior acurácia. Esse resultado foi consistente com o que encontramos em nossos experimentos. Já as redes Inception-V3 e ResNet 50 tiveram posições invertidas, porém com uma diferença de apenas 3%.

Rede Xception

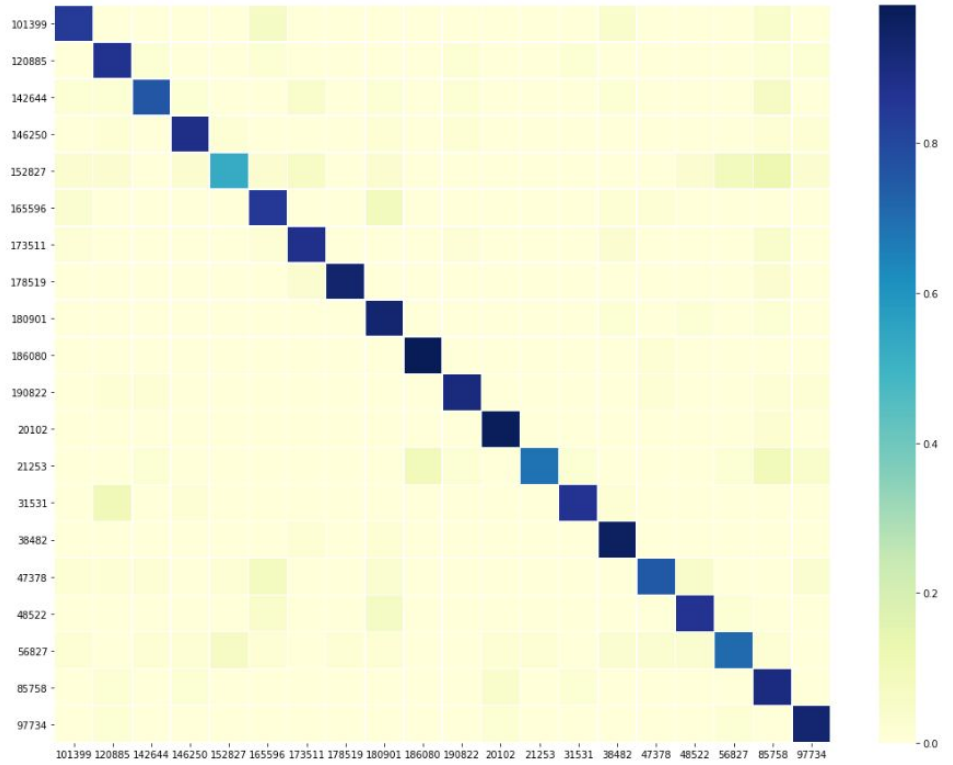
A rede Xception foi proposta por François Chollet em um artigo submetido em outubro de 2016 com o título “Xception: Deep Learning with Depthwise Separable Convolutions”. É uma rede que obtém um resultado melhor que a Inception-v3 utilizando o mesmo número de parâmetros.

Com essa rede, obtivemos os seguintes resultados:





Como essa rede teve o melhor desempenho nos nossos experimentos, aplicamos o modelo no nosso conjunto de teste, obtendo a matriz de confusão mostrada baixo. Vale observar temos uma diagonal muito clara e ótimos resultados ao considerarmos as 10 classes melhores classificadas.



Top 10 - teste

classes	acurácia
186080	0.9846
20102	0,9743
38482	0,9649
178519	0,9393
97734	0,9375
180901	0,9361
190822	0,9056
85758	0,9019
146250	0,8868
173511	0,8806

Acurácia de treino:

99.54%

Acurácia de validação:

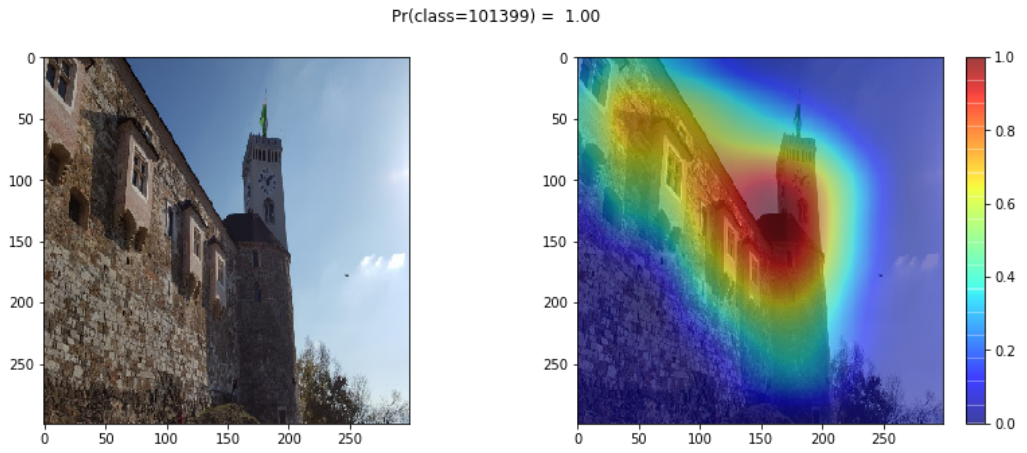
85.95%

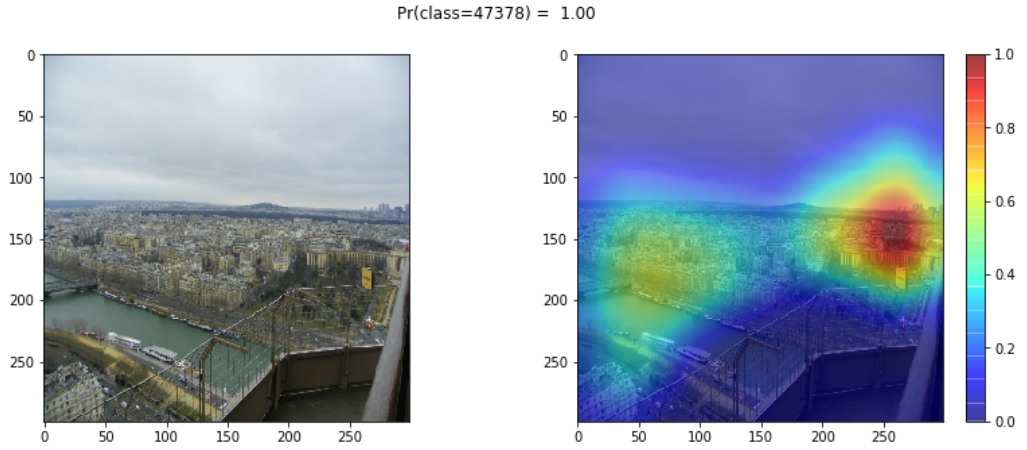
Acurácia de teste:

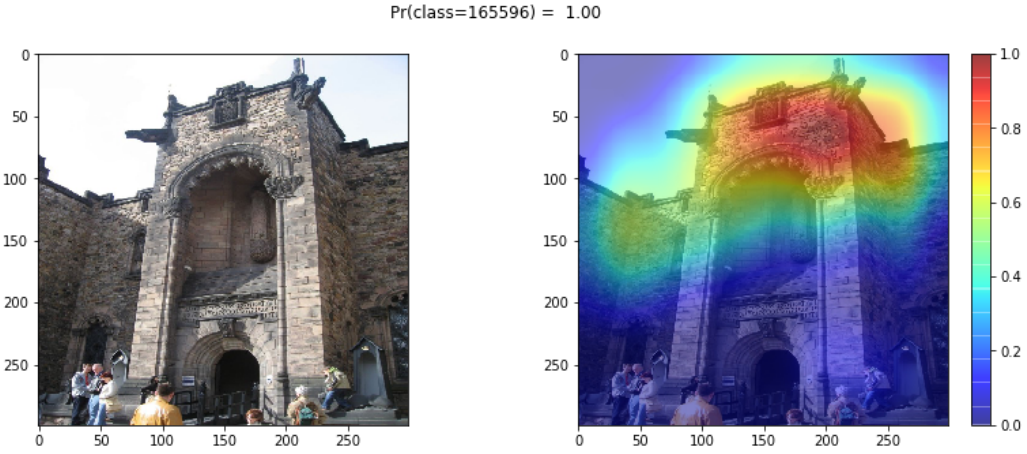
85.31%

Aprendizado da rede

Para tentar entender se o nosso modelo estava aprendendo características relevantes das imagens apresentadas, utilizamos a geração de mapas de ativação. Abaixo, temos 3 exemplos de mapas de ativação gerados para imagens classificadas corretamente.

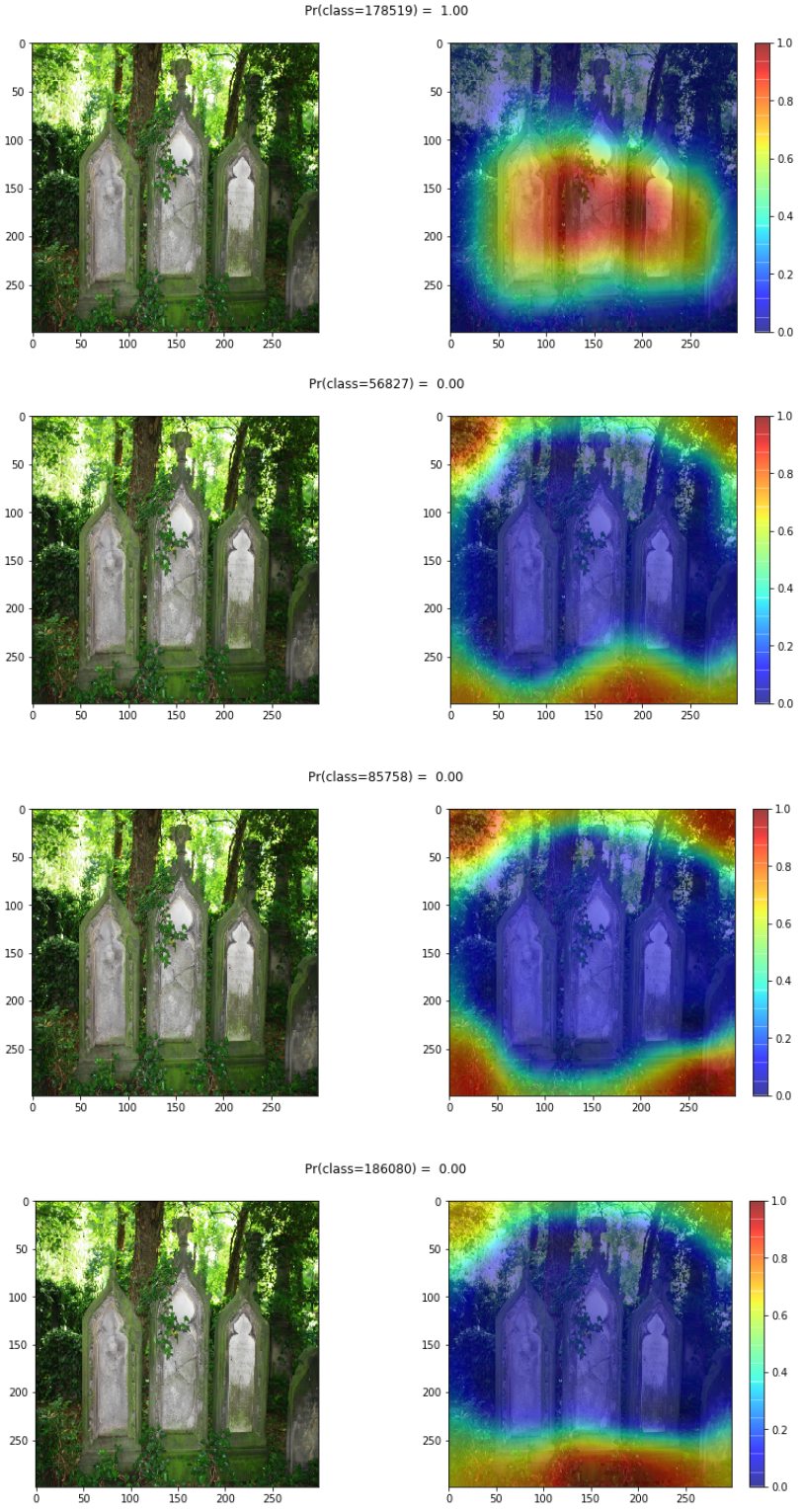






Podemos ver nesses mapas de ativação que a rede utiliza áreas visualmente interessantes para classificar as imagens fornecidas, mostrando um aprendizado relevante.

Ao verificar o mapa de ativação, de uma classes predita corretamente, olhando o resultado das top-4 melhores classificações, notamos que as ativações são diferentes demonstrando que a rede conseguir aprender algumas features relevantes nas imagens:



5. Conclusão

A realização desse projeto avançado nos permitiu aplicar técnicas aprendidas durante o curso em um problema de proporções muito maiores do que todos realizados em aula. Essa experiência nos mostrou que, além dos desafios inerentes a criação de modelos e busca por melhores acurácias nos resultados, existem uma série de desafios de engenharia que precisam ser vencidos para lidar com problemas de grandes volumes.

Baixar centenas de GB de datasets, processar milhões de imagens, lidar com epochs que levam horas para serem executadas, subir máquinas com múltiplas GPUs e centenas de GB de RAM a um custo astronômico ajudaram a clarear os desafios que as pessoas envolvidas nessa área precisam superar antes de entrar na parte de machine learning em si.

Outro ponto muito relevante que pudemos observar na prática é a coerência entre modelos estabelecidos e resultados de nossos experimentos. Obter melhores resultados com a rede Xception foi uma agradável surpresa, pois ela se tratava de uma aposta óbvia, dado os requisitos colocados pelo nosso time no início do trabalho.

Por fim, pudemos aprender, com experimentações, análise de erros, busca de informações e apoio dos monitores, como realizar um projeto de classificação de imagens. A oportunidade de realizar projetos de diferentes complexidades nessa mesma área nos mostrou que as dificuldades crescem quase que exponencialmente com o aumento do tamanho do desafio. Porém, tanto no caso do Fruits-360 quanto do Landmarks Recognition, ficou claro que as técnicas aprendidas no curso acabam sendo a base comum para o sucesso do projeto.

6. Referências

A seguir, algumas das referẽncias utilizadas para a realização deste trabalho.

Benchmark Analysis of Representative Deep Neural Network Architectures

https://www.researchgate.net/publication/328509150_Benchmark_Analysis_of_Representative_Deep_Neural_Network_Architectures

Review: Xception — With Depthwise Separable Convolution, Better Than Inception-v3 (Image Classification)

https://towardsdatascience.com/review-xception-with-depthwise-separable-convolution-better-than-inception-v3-image-dc967dd42568

Fine-tuning with Keras and Deep Learning

https://www.pyimagesearch.com/2019/06/03/fine-tuning-with-keras-and-deep-learning/

2D Global average pooling

https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/blocks/2d-global-average-pooling