Visão computacional com Redes Convolucionais

Classificação, busca, simillaridade, reutilização de extração de características

Ivan da Silva Brasílico

2019-07-18

Notas para a visualização em PDF

O site https://ivanbrasilico.github.io/model_ajna_1/ permite uma melhor navegação e visualização mais completa, incluindo cópia HTML de todos os notebooks.

O código-fonte completo do projeto está no GitHub:

https://github.com/IvanBrasilico/model_ajna_1

Visão geral

Nestes documentos estão centralizadas as anotações e histórico detalhado do treinamento e testes de alguns modelos de visão computacional.

Serão avaliadas as mesmas técnicas em bases diferentes, para efeito de comparação.

Linhas gerais:

- Baixar a base chestXRay
- Gerar e baixar base de Vazios e NcmsUnicos projeto AJNA
- Rodar treinamento e testes com modelo convolucional simples

- Rodar treinamento e testes com modelo "State of the Art" - Pesos do Imagenet
- Rodar treinamento e testes de extração de características para classificação
- Fazer teste de extração de características para busca de similaridade
- Rodar treinamento e testes usando redes siamesas
- Fazer teste de extração de características com redes siamesas para busca de similaridade
- Autoencoders, clusterização, para busca

Organização

- Proposta do projeto - Capstone Proposal
- Relatório resumido
- Relatório com detalhes da exploração de dados, modelos desenvolvidos, testes e iterações
- Conclusões finais
- Código Fonte do Projeto

O trabalho foi dividido em vários notebooks para melhor organização.

Estes notebooks estão com a seguinte nomenclatura

```
<número sequencial técnica/modelo><refinamento>-<descricao>-
<base>
```

Ex:

- 01 - número sequencial
- b - refinamento
- Transfer Learning - técnica
- vazios - base de dados

Assim:

01-RedeSimples-chestXRay é uma rede neural simples para classificar a base chestXRay

01-RedeSimples-vazios é uma rede neural simples para classificar a base vazios

01b-RedeSimples-vazios é a mesma rede/técnica do 01 mas com algumas modificações

Descrição do problema e das bases

Conforme detalhado em CapstoneProject, serão treinadas redes convolucionais simples do zero, modelos sofisticados com transfer learning, e redes siamesas. As bases utilizadas serão chestXRay, vazios e ncmsunicos.

Além da tarefa de classificação, o objetivo do projeto é tentar, com reaproveitamento, reutilizar artefatos obtidos em novas classificações e também validar o uso para agrupamento e similaridade. Assim, será possível economizar recurso computacional e humano em um ambiente de produção.

Adicionalmente, uma tendência atual da IA é a busca de "Sistemas de Inteligência Aumentada", ou IA "Centauro". Assim, os algoritmos são utilizados para empoderar operadores humanos. Com isso, além da classificação, prover agrupamento e busca de casos similares pode aumentar o poder de operadores humanos. Como exemplo, um médico pode procurar pacientes com casos similares para comparar prontuários e tratamentos, ou um analista de risco pode buscar imagens de escaneamento similares no rastro de uma fraude.

Métricas

Utilizando as definições de Andrew Ng em deeplearning.ai, primeiramente tentaremos definir um erro "aceitável". Para isso, será estimado o erro humano, em seguida será avaliado o erro de um modelo baseline e avaliados visualmente os erros cometidos.

Primeiramente será avaliada o acerto geral do modelo na base treinamento (*accuracy*), mas vigiando sempre a função custo (*loss*) e também os equivalentes na base validation (*val_loss* e *val_accuracy*). Assim, primeiramente

se terá como meta a redução do "bias evitável" e ter certeza de estar com um modelo promissor. Em seguida será avaliada a variância e sobreajuste, isso é, se o "gap" entre acc e val_acc é alto. Caso sejam, será avaliado se é um problema de sobreajuste ou um vício/erro nas bases de dados.

A *accuracy* (termo sem tradução exata para o Português exceto o neologismo acurácia, podendo ser traduzido também para exatidão) mede de todas as previsões realizadas, a porcentagem de acertos, isto é:

total de previsões corretas / total de previsões

Após esta primeira fase, passará a se olhar também outras métricas (precisão, *recall* e *f1-score*) de uma das classes ou das duas, conforme for mais importante para a visão de negócio.

A diferença das métricas precisão e *recall* para accuracy simples necessita entendimento do conceito de falso positivo e falso negativo. Assim, do total de exemplos da base submetidos ao modelo, podemos dividir por classe numa matriz de confusão. A matriz de confusão binária tem a seguinte configuração (considerando que fixamos POSITIVO como a classe 1):

```
                Valores reais
                Classe 0    Classe 1
Valores
preditivos
Classe 0        TN      FN

Classe 1        FP      TP
```

Assim:

TP - True Positive é a quantidade de exemplos que estão rotulados na classe 1 e foram classificados na classe 1 corretamente pelo modelo (classe 1 é, por exemplo, PNEUMONIA na nossa base chestXRay)

TN - True Negative é a quantidade de exemplos que estão rotulados na classe 0 e foram classificados na classe 0 corretamente pelo modelo (classe 0 é, por exemplo, NORMAL na nossa base chestXRay)

FP - False Positive é a quantidade de exemplos que estão rotulados na classe 0 e foram classificados na classe 1 incorretamente pelo modelo - seriam os "alarmes falsos", pessoas sem pneumonia que foram classificadas como contendo pneumonia.

FN - False Negative é a quantidade de exemplos que estão rotulados na classe 1 (Positivo para PNEUMONIA) e foram classificados na classe 0 incorretamente pelo modelo - seriam pessoas com pneumonia que o nosso modelo mandaria para caso, sem tratamento e com risco à sua saúde e até risco de morte.

Assim, seria melhor que nossos erros fossem concentrados no tipo FP - melhor mandar um paciente saudável para o médico revisar o exame do que mandar o paciente doente para casa sem o médico analisar mais a fundo. Precisamos minimizar o erro FN mesmo que isso custe diminuir um pouco o *accuracy*. A isso chamamos *Recall* ou recuperação: percentual do total de pacientes doentes detectados. Na tabela apresentada, matematicamente é TP / TP + FN. Se FN = 0 *Recall* é 100%. Precisão é a quantidade de acertos quando o modelo detecta positivo na classe 1, ou TP / TP + FP.

Note-se que caso se inverta a definição de "Positivo", o recall e precisão mudam, sendo quase trocados um pelo outro se a base é balanceada. Deve ser evitada esta confusão quando desta avaliação, por isso é importante deixar claro que a maximização de *recall* é para a detecção da classe 1 - PNEUMONIA.

Normalmente, todo modelo tem algum erro, e esse erro pode ser direcionado através de técnicas como Image Augmentation, peso de classes ou mudança de threshold. E há um *tradeoff* entre precisão e *recall*, quando um aumenta e outro diminui. Para medir o equilíbrio entre estas duas métricas, pode ser monitorado também o f1-score, que é a média harmônica entre Precisão e *Recall*

Como o projeto visa permitir reaproveitamento dos artefatos gerados e também gerar índices de similaridade para busca, serão avaliados também uso de disco, memória e velocidade de cada modelo.

Adicionalmente, para métrica de busca por similaridade, será utilizada avaliação visual pela escolha randômica de alguns exemplos e se o rótulo é coincidente para os primeiros resultados de uma busca por similaridade.

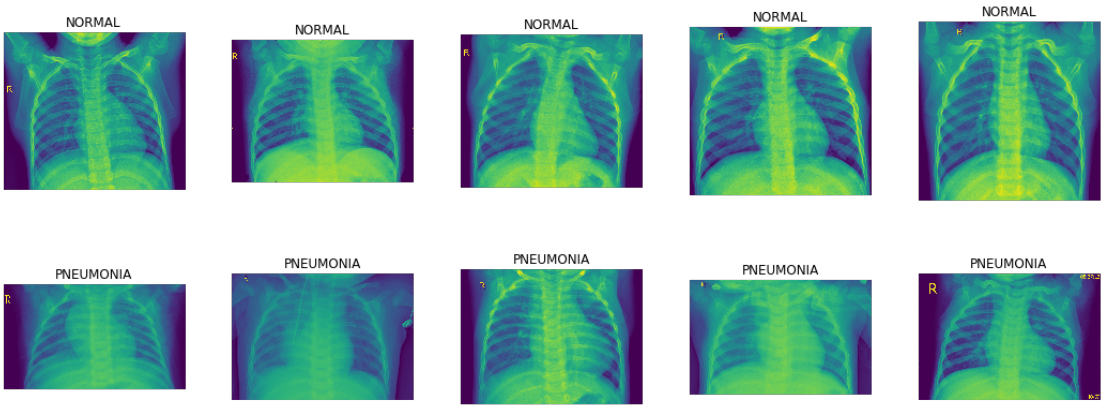
Exploração

BASE ChestXRay

A base chestXRay é composta de 5216 imagens na base de treinamento e 624 imagens na base teste.

São imagens de raio X de tórax, rotulados como paciente NORMAL e paciente com PNEUMONIA.

A base é levemente desbalanceada, havendo quase 3 vezes mais exemplos de pneumonia.



BASE Vazios

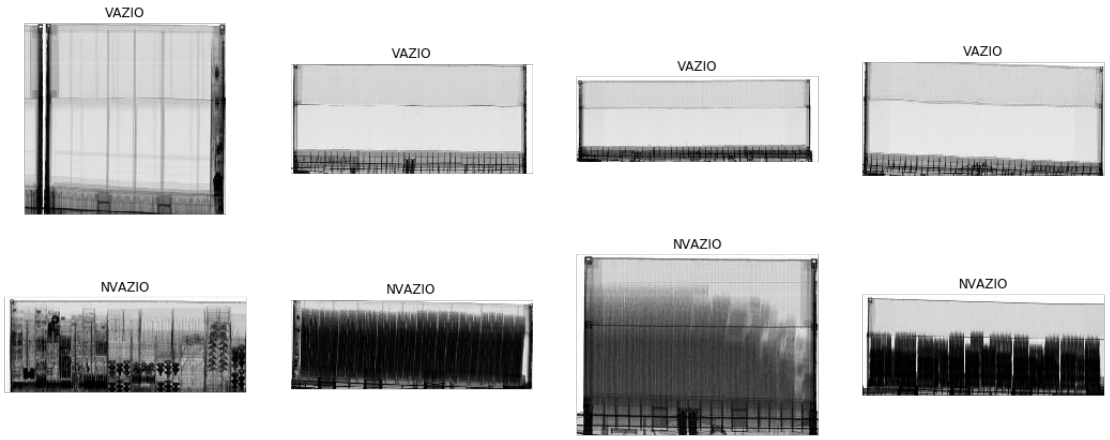
Esta base é composta por 20845 imagens de treinamento e 2317 imagens de validação. A base é balanceada.

São duas categorias: nvazio - contêineres contendo algum tipo de carga, mesmo que mínimo, e vazio - contêineres vazios.

Foram inseridos propositalmente, somando à extração aleatória, 3000 imagens de contêineres de "classificação difícil", imagens que algoritmos anteriores falharam para classificar.

Além disso, durante a exploração, foram descobertas em torno de 2,5% de imagens rotuladas erradamente e 2% de imagens que mesmo a visão humana teria dificuldade de saber se está vazio ou não.

Assim, como o melhor desempenho obtido foi próximo de 98% para base "limpa" e de 96% para base completa (ver detalhes no relatório detalhado e nos respectivos notebooks) pode ser considerado que para esta tarefa foi obtido um classificador excelente.



BASE NCMs únicos

Esta base é composta de 41809 imagens de 868 categorias.

São imagens de inspeção não invasiva de contêineres.

Benchmark

O primeiro modelo a ser treinado será sempre uma rede convolucional bem simples.

Além disso, na base Vazios, há um modelo em produção, uma SVM, que poderá ser comparada.

As redes neurais convolucionais são hoje o "estado da arte" em visão computacional. As camadas convolucionais aprendem a aplicar filtros em diversas regiões das imagens, destacando formas, texturas, linhas, de acordo

com a necessidade da tarefa em que estão sendo treinadas. As convoluções são aplicadas consecutivamente, em objetos cada vez maiores, porque as imagens são progressivamente filtradas por convuluções, ou mais comumente por camadas de *pooling* que diminuem o tamanho da entrada, destacando partes mais importantes, e os filtros das convoluções mais profundas combinam então os destaques dos filtros predecessores. Nas últimas camadas, estes mapas de características descobertos pelas convoluções são combinados em uma rede neural convencional conectada para utilização na tarefa de classificação.

Como a rede neural é um aproximador de funções universal, através de *backpropagation* e gradiente descendente esta consegue derivar os pesos necessários para a tarefa de classificação treinada, desde que adequada projetada e treinada.

Antes do advento das redes neurais, os algoritmos que costumavam obter melhores resultados em classificação de imagens eram as Suport Vector Machines - SVMs. As SVMs podem utilizar diversos "kernels" para a tarefa de classificação, tendo sido o *kernel* RBF utilizado com sucesso durante décadas, antes da rede AlexNet ter baixado em mais de 10 pontos percentuais o erro no desafio AlexNet, antes dominado por SVMs. O kernel rbf funciona procurando várias funções gaussianas, em cada dimensão, que separem por uma margem específica, que é um hiperparâmetro do kernel, a maior quantidade de exemplos das classes.

Assim, os baselines utilizados serão um kernel RBF já em produção e uma rede neural extremamente simples e rápida para treinar. Em seguida, será utilizada a rede DenseNet121 que apresenta um bom equilíbrio entre resultados comprovados em bases difíceis, consumo de memória e complexidade computacional, com a técnica de *Transfer Learning*. Em seguida, será treinado um modelo de rede siamesa. Todos os modelos serão avaliados paralelamente conforme seção métricas.

Metodologia

Pré processamento de dados

Está sendo utilizado o pacote PIL ou o ImageDataGenerator(que usa pacote PIL) do keras para abertura das imagens e redimensionamento com ANTALIAS. Os valores RGB originais estão sendo reescalados dividindo por 255. Além disso foram testadas diversas opções de Image Augmentation.

No caso da base Vazios, foi detectado erro de rotulagem e automaticamente gerada uma base filtrada através de threshold em um dos classificadores treinados.

Os detalhes do pré processamento e principalmente da execução estão nos Notebooks. Resumo/índice no próximo item.

Análise

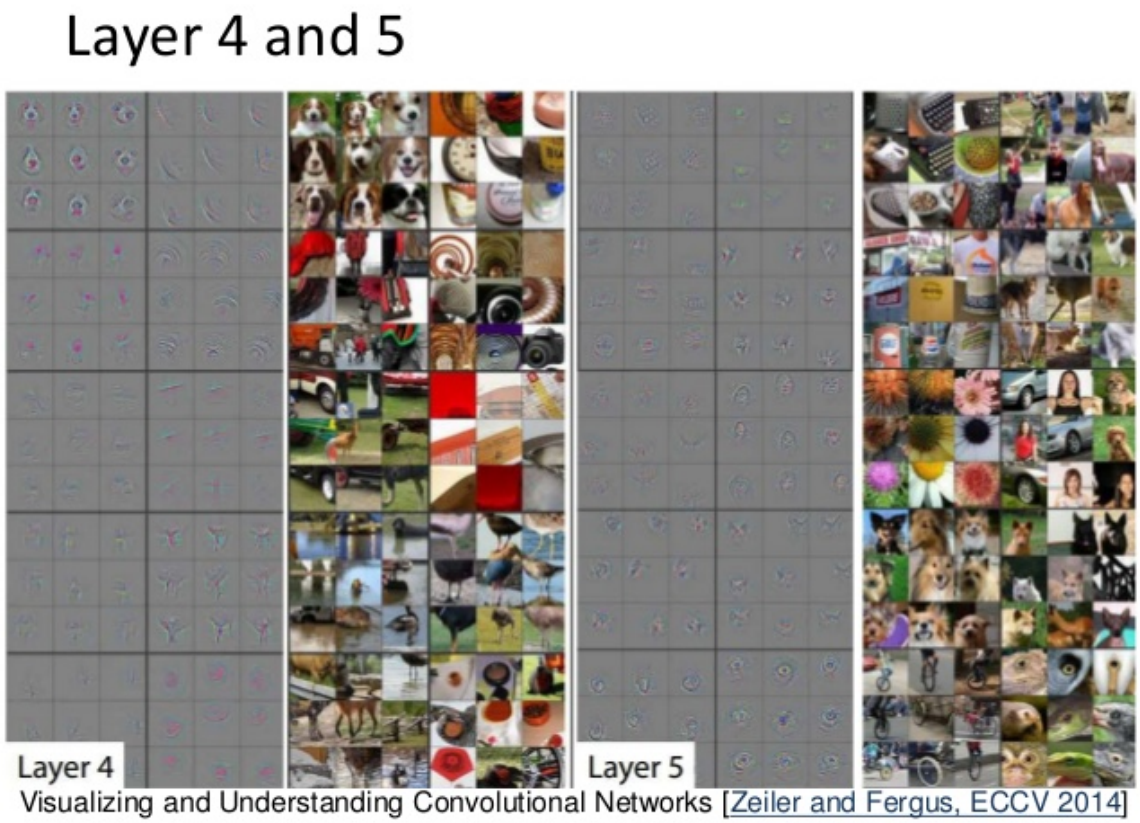
Para obter melhores resultados, serão provavelmente necessárias a utilização de técnicas adicionais, conforme citado anteriormente:

Tranfer Learning:

Consiste em utilizar pesos que representam conhecimento acumulado/ aprendido por uma rede neural em domínio similar. Assim, existem redes que são referências em *papers* publicados, representando avanços tecnológicos, e que foram treinadas em milhões de imagens, como na base ImageNet (http:// www.image-net.org/).

Estas redes são treinadas por semanas, nestes milhões de imagens, aprendendo a "ver" formas, linhas, texturas, etc. Através do uso das primeiras camadas destas redes, excluindo apenas as últimas, responsáveis pela classificação, podemos extrair um resumo de todo esse aprendizado e transferir

este conhecimento para uma nova tarefa.



Visualizing and Understanding Convolutional Networks [Zeiler and Fergus, ECCV 2014]

Image Augmentation:

Como temos relativamente poucos exemplos, uma forma de melhorar o aprendizado e principalmente evitar sobreajuste é aplicar pequenas modificações aleatóriamente na base de treinamento. Assim, simulamos o treinamento com uma base maior.

O keras possui diversas funções já pré preparadas para conseguir realizar esta tarefa com facilidade (https://keras.io/preprocessing/image/).

Podem ser aplicados zooms, recortes, mudança de brilho, espelhamento, pequenos deslocamentos laterais, entre outros. Deve se ter em mente o domínio do problema, para não realizar tranformações que prejudiquem o aprendizado. Por exemplo, se o objeto a ser detectado tem um tamanho ou posição fixos, usar recorte ou zoom seria inadequado e mesmo os deslocamentos não podem ser grandes.

Siamese networks:

As redes siamesas são em tudo iguais às redes convolucionais normais, exceto por uma particularidade: você sempre passa duas imagens pela mesma rede e no final compara as saídas. Com isso, a rede aprende uma função de similaridade, podendo ser utilizada para reconhecer objetos que não estavam na base de treinamento, desde que pertençam ao mesmo domíno. Junto com as GANs, são um avanço interesssante sobre as redes neurais convencionais, podendo ajudar a resolver a imensa fome por dados para treinamento. Mas, assim como as GANs, são um pouco mais difíceis de treinar e utilizar.

Como aprendem uma função de similaridade, redes siamesas podem aprender um melhor *embedding* da imagem (os números que estarão na última camada e representam tudo que foi "visto" pelas camadas convolucionais) e podem ser utilizadas para buscas de similaridade, One Shot Learning,
para lidar com classes desbalanceadas e até para fazer acompanhamento visual de objetos (object tracking).

One Shot Learning

Signature Verification

Implementação e refinamento

Ver próximo item, que resume os achados de cada notebook utilizado.

Relatórios da BASE ChestXRay

01-Baseline-redesimples-chestXRay

01-Baseline-redesimples-chestXRay

Rede convolucional bem simples treinada do zero.

Input shape = 150, 150

acc: 0.9279 - val_acc: 0.8285

01b-Baseline-redesimples-chestXRay-tamanhomaior

01b-Baseline-redesimples-chestXRay-tamanhomaior

Rede convolucional bem simples treinada do zero. Treinamento em 04/09/2019:

Foram realizadas várias rodadas(sempre continuando pesos do menor val_loss anterior):

- A primeira com ImageAugmentation e lr=0.001, melhor acc=0.94 e melhor val_acc=0.82 Mesmo a rede sendo simples, aparenta ligeiro overfitting
- A segunda com lr=0.0001 e mais épocas para os callbacks, melhor acc=0.94 e melhor val_acc=0.83
- A terceira sem ImageAugmentation, com lr muito pequena. Embora ImageAugmentation seja uma técnica para reduzir overfitting, e a priori tirar possa parecer contrasenso, apenas para
  testar se deixar a base de treinamento mais parecida com a de testes reduz erro de generalização, ao menos nesses exemplos e no "fine tunning"

Conforme previsto pela teoria, o sobreajuste aumentou.

acc foi para 0.96 e val_acc caiu para menos de 0.80

- Quarta tentativa, com regularização L1 e L2 na última camada e otimizador Adam, pareceu que ia conseguir melhoria, foi expandido o treinamento para 50 épocas iniciando com uma lr maior, mas a melhoria foi apenas marginal, com val_acc ensaiando ultrapassar 0.87 mas oscilando bastante

Em 04/06/2019 o melhor modelo foi:

Epoch 14/50 acc: 0.9507 val_acc: 0.8429

Conclusões/próximos passos

- Tentar aumentar regularização, utilizar keras-tuner
- Testar modelo pré-treinado mais poderoso (TransferLearning)
- Olhar exemplos de kernel no kaggle com melhor desempenho em busca de idéias

02c-TransferLearningSimples-FeatureExtractionRegularizer-chestXRay

02c-TransferLearningSimplesFeatureExtractionRegularizer-chestXRay

Utilizar DenseNet121 como feature extraction. Treinar classificador na saída desta rede.

Resultado testes: acc: 0.93 val_acc: 0.82

Próximo passo:

Gravar em .npy uma matriz com todas as features extraídas da base de treinamento e fazer Grid Search e Random Search do melhor classificador obtido.

02d-TransferLearning-FeatureExtraction-HyperParamTuner-chestXRay

02d-TransferLearningFeatureExtractionHyperParamTuner-chestXRay

Esta rede usa como entrada uma última camada maxpooling já salva, de saída da DenseNet121 aplicada à base de treinamento. Como todo o processamento convolucional já está realizado, o treinamento do classificador é centenas de vezes mais rápido. Assim, facilita o tunning da camada classificadora.

Resultado:

Foi possível obter um classificador utilizando somente a saída da DenseNet121 original com pesos da imagenet:

Base original: acc 0.95 val_acc 0.89 recall pneumonia: 0.95 0.97

02e-auxiliar-ImageAugmentation

02e-auxiliar-ImageAugmentation

Este notebook é apenas para gerar uma base aumentada pré-processada. Será utilizado pelo outro notebook 02e.

O objetivo é tentar diminuir o sobreajuste / distância entre acc e val_acc e agilizar a fase de treinamento.

02e-FineTunning-chestXRay

02e-FineTunning-chestXRay

Aqui está sendo treinada uma rede DenseNet121 do 02c empilhada com o classificador do 02d.

Problemas: não ficou claro se os pesos do notebook 02d foram aproveitados. Eles são carregados, os testes dão resultado similar ao 02d, mas quando inicia o treinamento de fine tunning os números de acc e val_acc caem próximos de

0.5, para depois voltarem a subir, mesmo quando se utiliza uma lr extremamente baixa.

Melhor modelo: Transfermodelweights02e_etapa2.02-0.66.hdf5

Base aumentada: acc 0.99 val_acc 0.83

Obs: Houve um problema, o acc na base train indica 99% no treinamento, mas estranhamente cai para 95% no relatório. Investigar.

Base original: acc 0.95 val_acc 0.89 recall pneumonia: 0.96 0.97

03-Busca-TransferLearning-Imagenet-chestXRay

03-Busca-TransferLearning-Imagenet-chestXRay

Teste do uso das features extraídas de uma rede pré-treinada como hash para busca de similaridade.

Testar através de distância euclidiana se a última camada de rede neural DenseNet121 possui informação interessante para possibilitar busca por similaridade.

Foram rodadas 1.000 simulações aleatórias de busca para vários batchs diferentes, de 512 itens para base train e 256 itens para base. No final foram rodadas 1.000 simulações para 10 batches da base treinamento.

A avaliação foi realizada por coincidência de classe nos primeiros 10 e 20 itens e também foi realizada avaliação visual interativa.

A avaliação visual é muito difícil, precisaria de um especialista médico para avaliar.

A avaliação por classe deu uma coincidência média de menos de 80%, considerada insuficiente. Também foi extraída a estatística por classe:

0 = NORMAL 1 = PNEUMONIA

Resultados utilizando MaxPooling

- Acerto classe 0: 53959 de 72780 (0.74)
- Acerto classe 1: 103373 de 127220 (0.81)

Resultados utilizando AvgPooling

- Acerto classe 0: 55893 de 75900 (0.74)
- Acerto classe 1: 103782 de 124100 (0.84)

Assim, as *features* extraídas da rede treinada na ImageNet se mostraram insuficientes para busca. Não obstante, podem ser um ponto de partida, para treinamento de autoencoders ou outras funções para gerar um hash para busca de similaridade.

Observações finais

Considerando que para este tipo de problema o mais importante é um recall alto para pneumonia.

O modelo final tem um recall excelente, embora o desejável neste caso seja 100%, não sabemos se há erro de rotulagem nem qual o erro humano, muito menos o Bayes Error. Portanto, não dá para saber se é factível melhorar acima de 95-97% de recall.

Não foi possível obter ganhos significativos em relação ao baseline com as técnicas empregadas. A melhoria foi marginal, de menos de 5% em relação à rede neural simples. Tabela abaixo.

REDE 01b Accuracy: acc 0.95 val_acc 0.85 recall pneumonia: 0.94 0.95 REDE 02e Accuracy: acc 0.95 val_acc 0.89 recall pneumonia: 0.96 0.97

A diferença entre treinamento e validação demonstra uma variância grande, mas pelos testes na base seria importante checar se não se trata de um *data mismatch*. Esta base parece ter problemas de balanceamento e também de distribuição. Como próximo passo, seria interessante fundir todos os exemplos da base orginal (train, val, test) em uma base única e fazer um *resample* das bases de treinamento e validação, rodando cópias destes notebooks e

comparando os resultados. Além disso, testar técnicas adicionais de *image augmentation* e balanceamento de classes (parâmetro *class weight* ou aumento de uma categoria).

Relatórios da BASE Vazios

01-Baseline-redesimples-vazio

01-Baseline-redesimples-vazio

Rede convolucional bem simples treinada do zero.

acc: 0.9551 - val_acc: 0.9564

Este notebook também contém visualizações para tentar entender melhor o que foi aprendido pela rede.

01b-Baseline-redesimples-vazio-tamanhomaior

01b-Baseline-redesimples-vazio-tamanhomaior

Mesma rede convolucional, mas treinada com entrada maior (224x224). O tamanho de entrada é o mesmo da maioria dos modelos treinados na imagenet.

acc: 0.9589 - val_acc: 0.9616

Em 26/06/2019:

Rodada três vezes a sequência acima, 99, 101 e 103 erros de classificação (a mudança é devido a técnicas de image augmentation). Precisão de 100% na classe 0 e recall 91% ou seja 9% de erros tipo II falso negativo (predição 1 rótulo 0).

Analisando visualmente o diretório, pelo menos 25% dos erros são de rotulagem (os contêineres realmente não contém carga. Dos 70-75 erros restantes, em 20% do total o contêiner está escuro, parecendo ter carga de espuma. Em torno de 30% do total também há diversos tipos de ruídos na imagem, desde carretas que invadem a área do contêiner até borrões laterais na imagem, mas não carga. Então também é contêiner efetivamente vazio. Nos erros restantes (apenas 20% de 9%) parece haver erro de classificação, mas o contêiner contém pouca carga.

Conclusões:

```
* O erro real do algoritmo pode ser de apenas 2-4% e apenas na
classe Não Vazio.
Este erro poderia ser melhorado com melhora no recorte do
contêiner e na limpeza da imagem original.
* Dos 9% de erros, 2% são aparentemente "fraudes": contêineres
não continham carga
* Dos 9% de erros, 2% podem ser "fraude" ou falha no escâner
* Necessário proibir carretas que obstruam o contêiner
```

O algoritmo está tentendo a ignorar cargas de contêineres declarados como vazios mas borrados/sujos ou com muito pouca carga ou com carga uniforme de espumas/materias pouco densos. Talvez fosse interessante forçar o algoritmo a ser mais tendente a diminuir este erro, mesmo que isto custasse aumento de falso positivo na classe vazio.

01b2-Baseline-redesimples-vazio-tamanhomaior-augmented-filtered

01b2-Baseline-redesimples-vazio-tamanhomaior-augmented-filtered

Este notebook aplica o mesmo método que 01b, mas trocando para base aumentada e filtrada (redução de erros de rótulo) produzida por 02c e o2d2, isto é, foi gerada nova base, já aumentada e excluindo erros acima e abaixo de um threshold do classificador 02c, que na inspeção visual ficou evidente tratarem-se de erros de rotulagem, isto é, data mismatch.

Base aumentada: acc: 0.97 - val_acc: 0.97 Base original: acc: 0.96 - val_acc: 0.96

01b3-Baseline-redesimples-vazio-tamanhomaior-augmented-filtered-menostranform

01b3-Baseline-redesimples-vazio-tamanhomaior-augmented-filtered-menostransform

Este notebook aplica o mesmo método que 01b, mas trocando para base aumentada e filtrada (redução de erros de rótulo) produzida por 02c e o2d2, isto é, foi gerada nova base, já aumentada e excluindo erros acima e abaixo de um threshold do classificador 02c, que na inspeção visual ficou evidente tratarem-se de erros de rotulagem, isto é, data mismatch.

Além disso, na inspeção visual do notebook 01b2 ficou a impressão de que os erros que ainda estavam ocorrendo eram: erros que mesmo o humano teria dificuldade (contêineres com espuma, por exemplo) ou erros de rótulo persistentes. Além desses, o algoritmo ainda erra em alguns poucos casos de contêiner contendo muito pouca carga, especialmente se esta se concentra apenas no solo (provavelmente confunde com imagens de vazio com solo poluído por carretas) ou somente em uma das portas (provavelmente confundindo com reefer). Assim, neste notebook foi diminuída a amplitude das transformações de imagem aumentada para checar o resultado.

Base aumentada: acc: 0.97 - val_acc: 0.98 Base original: acc: 0.96 - val_acc: 0.96

02-TransferLearningSimples-vazio

02-TransferLearningSimples-vazio

Rede Densenet121, pré treinada na imagenet.

acc: 0.9545 - val_acc: 0.7126

Claramente, houve um sobreajuste muito grande. Os erros de classificação cometidos são gritantes.

Foi realizado fine tunning do último bloco convolucional (conv5):

acc: 0.9523 - val_acc: 0.8045

Apesar dos resultados ruins na generalização, necessário explorar mais esta possibilidade. A dificuldade pode ser devido ao bias em textura da imagenet. Note-se que esta base é em tons de cinza, e o mais importante é a geometria. Imagenet é colorida e textura é importante.

ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness https://arxiv.org/abs/1811.12231

02b-TransferLearningSimplesRegularizer-vazio

02b-TransferLearningSimplesRegularizer-vazio

Rede Densenet121, pré treinada na imagenet, com regularização.

02c-TransferLearning-FeatureExtractionRegularizer-vazio

02c-TransferLearningSimplesFeatureExtractionRegularizer-vazio

Rede Densenet121, pré treinada na imagenet, com regularização.

acc: 0.9408 - val_acc: 0.9514

Neste caso, se optou por utilizar as camadas pré treinadas para feature extraction, e, foi utilizada Max Pooling na última camada em vez de Avg Pooling.

Observações:

Após a extração das features das imagens, o treinamento do classificador é **centenas de vezes** mais rápido. Assim, a extração separada dos features permitirá treinar vários classificadores, fazer grid search e cross validation, entre outros.

Conforme demonstrado acima, há entre as imagens da classe nvazio diversos exemplos que parecem da classe vazio. Ou são erros de base ou são exemplos extremamente similares aos vazios. O aprendizado deve melhorar eliminando estes da base. Será criada uma cópia da base sem esses exemplos, para testar os mesmos algoritmos e comparar.

02c2-TransferLearningFeatureExtraction-Vazio

02c2-TransferLearningFeatureExtraction-Vazio

- Extrair features para numpy com imageaugmented bem "suave" (teste 01b3) produzida por 02c e o2d2
- Rodar com maxpool e com avgpool para poder comparar
- Rodar keras_tuner e comparar resultados com melhor resultado da rede simples

Base original maxpool: acc: 0.9604 - val_acc: 0.9566 Base original avgpool: acc: 0.9594 - val_acc: 0.9588

Parece que não importa o que se tente, há um platô em torno de 0.96 para accuracy na base original.

Com a base "limpa" de alguns erros de rotulagem, foi possível subir este platô para um pouco mais de 97%. Como a maioria dos erros é na classe vazio, antes de prosseguir: * Testar neste mesmo notebook treinamento com class_weigth * Copiar este notebook e repetir mesmos passos na base gerada por 02d2

O uso de class_weight 3 para a classe 0 (não vazio) causou queda marginal na accuracy total, mas distribuindo melhor os erros, conforme tabela abaixo (a accuracy caiu nas casas centesimais, em torno de 4 centésimos):

BASE TEST

Sem class_weight

```
              precision    recall  f1-score   support

         0.0       0.99      0.92      0.96      1166
         1.0       0.93      0.99      0.96      1138
```

Com class_weight

```
              precision    recall  f1-score   support

         0.0       0.97      0.94      0.95      1166
         1.0       0.94      0.97      0.95      1138
```

BASE TRAIN

Sem class_weight

```
              precision    recall  f1-score   support

         0.0       1.00      0.93      0.96     10494
         1.0       0.93      1.00      0.96     10306
```

Com class_weight

```
              precision    recall  f1-score   support

         0.0       0.98      0.95      0.96     10494
         1.0       0.95      0.98      0.96     10306
```

02c3-TransferLearningFeatureExtraction-Vazio

02c2-TransferLearningFeatureExtraction-Vazio

- Extrair features para numpy com imageaugmented bem "suave" e filtrado (mesma base que notebook 01b3)
- Rodar com maxpool e com avgpool para poder comparar
- Rodar keras_tuner e comparar resultados com melhor resultado da rede simples

Detalhes no notebook. Resumindo, os resultados foram muito similares ao notebook 01b3:

- aumento de 2% em accuracy em relação à base original, provavelmente pela correção de erros de rótulo
- De resto, resultados similares ao notebook 02c2, em todas as tabelas (com o aumento de quase 2%)

02d-auxiliar-ImageAugmentation-Vazios

02d-auxiliar-ImageAugmentation-Vazios

Notebook auxiliar para gerar uma base aumentada.

02d2-auxiliar-ImageAugmentationMenosTransfom-Vazios

02d2-auxiliar-ImageAugmentationMenosTransfom-Vazios

Notebook auxiliar para gerar uma base aumentada com poucas transformações.

02d2-auxiliar-ImageAugmentationMenosTransfom-Vazios

02d2-auxiliar-ImageAugmentationMenosTransfom-Vazios

Notebook auxiliar para gerar uma base aumentada com poucas transformações.

03-Busca-TransferLearning-Imagenet-Vazios.ipynb

03-Busca-TransferLearning-Imagenet-Vazios

Teste do uso das features extraídas de uma rede pré-treinada como hash para busca de similaridade.

Métricas utilizadas:

- Dos 10 primeiros e dos 20 primeiros resultados(de um total de 512), quantos pertencem à mesma classe?

Foram rodadas 1.000 simulações aleatórias de busca para vários batchs diferentes, de 512 itens para base train e 256 itens para base. No final foram rodadas 1.000 simulações para 10 batches da base treinamento.

A avaliação foi realizada por coincidência de classe nos primeiros 10 e 20 itens e também foi realizada avaliação visual interativa.

A avaliação visual demonstrou precisão alta na comparação de vazios. Mas a comparação de contêineres com Carga, imagem com mais informação, pareceu bem mais prejudicada.

A avaliação por classe deu uma coincidência de pouco mais de 80%, considerada insuficiente. Também foi extraída a estatística por classe:

0 = Não vazio 1 = Vazio

Resultados utilizando MaxPooling

- Acerto classe 0: 70719 de 99920 (0.71)
- Acerto classe 1: 85712 de 100080 (0.86)

Resultados utilizando AvgPooling

- Acerto classe 0: 79105 de 107680 (0.73)
- Acerto classe 1: 83533 de 92320 (0.90)

Assim, as *features* extraídas da rede treinada na ImageNet se mostraram insuficientes para busca. Não obstante, podem ser um ponto de partida, para treinamento de autoencoders ou outras funções para gerar um hash para busca de similaridade.

Observações

Os resultados da rede simples treinada do zero foram similares ao uso de rede DenseNet, mas a extração de features com rede pré treinada na imagenet pode ser um método universal base para vários classificadores, buscas e análises.

Assim, quando uma imagem entrar no Banco de Dados, pré extrair as features via uma rede pré treinada, salvando no Banco de Dados, pode servir como ponto de entrada para vários tipos de classificadores e comparações, salvando memória e processamento posterior.

Os resultados utilizando maxpool e avgpool como extrator de características foram muito similares, com leve vantagem para avgpool nos resultados e menor tempo de convergência.

Os melhores resultados obtidos foram de 96% de accuracy e 96% de f1-score, sendo que a base parece ter em torno de 2% de erros de rotulagem. Com a base limpa, o resultado subiu a quase 98%. Embora pela visualização haja espaço para melhora (alguns contêineres não vazios com muito pouca carga mas facilmente identifiáveis pelo olho humano classificados como vazios), o modelo está muito próximo de um candidato a colocação em produção. Outro ponto interessante é que foi demostrado ser possível utilizar um classificador extremamente simples e rápido, que utiliza como ponto de entrada apenas 1024 números que podem ser pré-extraídos das imagens pela rede DenseNet121 e ocupa apenas 14MB de RAM por batch.

```
           precision    recall  f1-score   support

      0.0       0.98      0.95      0.96     10494
      1.0       0.95      0.98      0.96     10306
```

Conclusões

Observações

Foram utilizadas várias técnicas e realizadas diversas iterações/melhorias dentro das técnicas, sucessivamente.

Além disso, paralelamente, testes parecidos foram realizados em bases diferentes para testar generalidade do modelo.

Base ChestXRay

Conforme era esperado, esta base se mostrou mais difícil de trabalhar do que a base de escaneamento de contêineres para classificar vazio/ não vazio

Considerou-se que para este tipo de problema o mais importante é um recall alto para pneumonia.

O modelo final tem um recall excelente, embora o desejável neste caso seja 100%, não sabemos se há erro de rotulagem nem qual o erro humano, muito menos o Bayes Error. Portanto, não dá para saber se é factível melhorar acima de 95-97% de recall.

Não foi possível obter ganhos significativos em relação ao baseline com as técnicas empregadas. A melhoria foi marginal, de menos de 5% em relação à rede neural simples. Tabela abaixo.

REDE 01b Accuracy: acc 0.95 val_acc 0.85 recall pneumonia: 0.94 0.95 REDE 02e Accuracy: acc 0.95 val_acc 0.89 recall pneumonia: 0.96 0.97

A diferença entre treinamento e validação demonstra uma variância grande, mas pelos testes na base seria importante checar se não se trata de um *data mismatch*.

Esta base parece ter problemas de balanceamento e também de distribuição. Como próximo passo, seria interessante fundir todos os exemplos da base

orginal (train, val, test) em uma base única e fazer um *resample* das bases de treinamento e validação, rodando cópias destes notebooks e comparando os resultados. Além disso, testar técnicas adicionais de *image augmentation* e balanceamento de classes (parâmetro *class weight* ou aumento de uma categoria).

Base Vazios

Nesta base também considerou-se que o mais importante seria obter um recall alto, reconhecendo principalmente contêineres declarados como vazios mas contendo carga.

Os resultados da rede simples treinada do zero foram similares ao uso de rede DenseNet, mas a extração de features com rede pré treinada na imagenet pode ser um método universal base para vários classificadores, buscas e análises.

Assim, quando uma imagem entrar no Banco de Dados, pré extrair as features via uma rede pré treinada, salvando no Banco de Dados, pode servir como ponto de entrada para vários tipos de classificadores e comparações, salvando memória e processamento posterior.

Os resultados utilizando maxpool e avgpool como extrator de características foram muito similares, com leve vantagem para avgpool nos resultados e menor tempo de convergência.

Os melhores resultados obtidos foram de 96% de accuracy e 96% de f1-score, sendo que a base parece ter em torno de 2% de erros de rotulagem. Com a base limpa, o resultado subiu a quase 98%. Embora pela visualização haja espaço para melhora (alguns contêineres não vazios com muito pouca carga mas facilmente identifiáveis pelo olho humano classificados como vazios), o modelo está muito próximo de um candidato a colocação em produção. Outro ponto interessante é que foi demostrado ser possível utilizar um classificador extremamente simples e rápido, que utiliza como ponto de entrada apenas 1024 números que podem ser pré-extraídos das imagens pela rede DenseNet121 e ocupa apenas 14MB de RAM por batch.

Neste problema, a rede Siamesa treinada também apresentou resultados excelentes. Utilizar redes siamesas para classificação adiciona uma complexidade: como a rede sempre compara duas imagens, não há uma rede treinada para simplesmente fazer a classificação da imagem, mas sim dar um número (próximo de zero para itens iguais ou da mesma classe, próximo de 1 para itens diferentes). Optou-se por comparar a imagem a ser classificada com duas imagens: uma da classe 0 e outra da classe1. A que retornar menor número é considerada a correta.

Baseline SVM

```
         precision    recall  f1-score   support

      0     1.0000    0.9117    0.9538      1166
      1     0.9179    1.0000    0.9572      1151
```

Rede 01b3 (baseline simples)

```
         precision    recall  f1-score   support

      0     1.0000    0.9245    0.9608      1166
      1     0.9290    1.0000    0.9632      1151
```

Rede 02c3 (*Features* da DenseNet121 treinada na Imagenet)

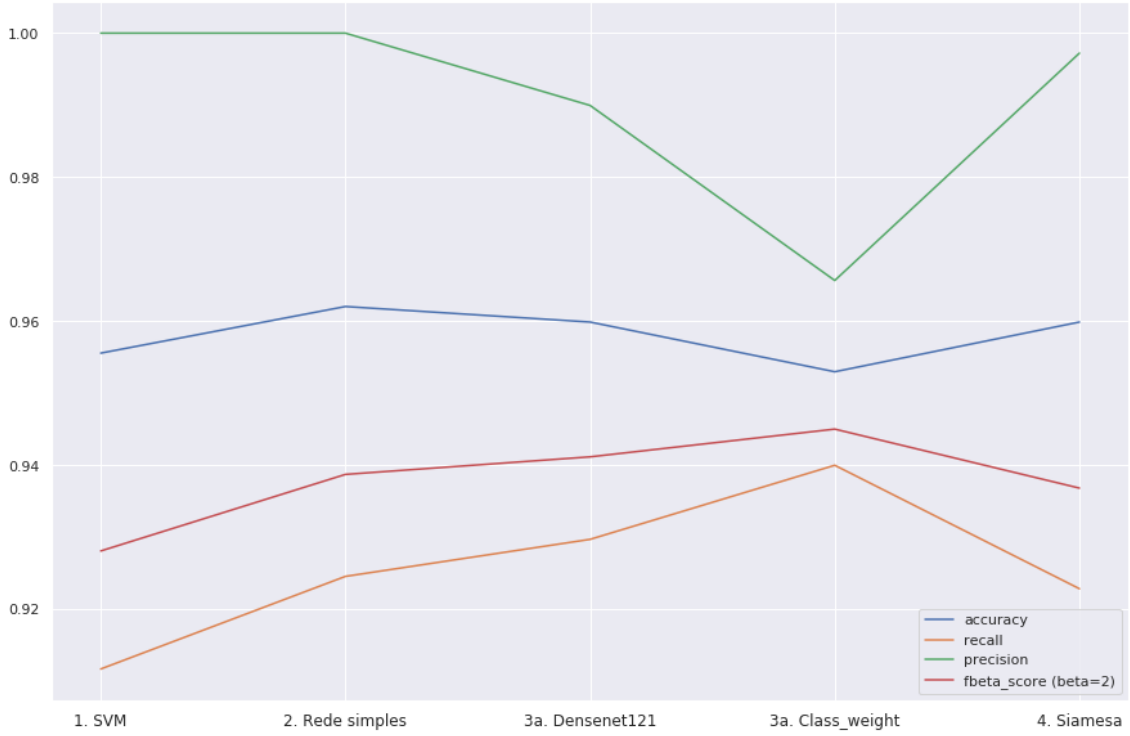
```
         precision    recall  f1-score   support

      0     0.9900    0.9297    0.9589      1166
      1     0.9329    0.9904    0.9608      1151
```

Rede Siamesa

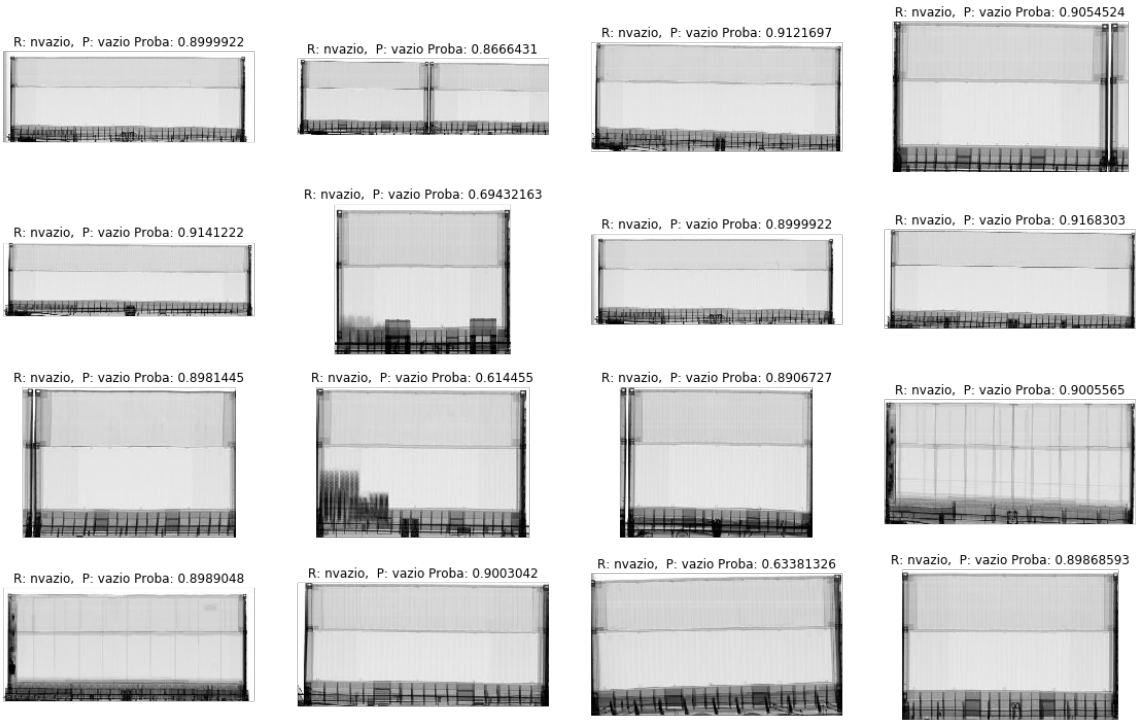
```
       precision    recall  f1-score   support

    0.0     0.9972    0.9228    0.9586      1166
    1.0     0.9273    0.9974    0.9611      1151
```



Mais do que apenas ver os números, seria interessante visualizar e entender os erros que cada modelo está cometendo. Assim, no notebook 05 comparei também os erros de cada modelo entre si. A divergência maior foi entre o modelo 1 e os demais, chegando a 3%. Os modelos de redes neurais possuem menos de 1,5% de divergência entre os erros e acertos. Isso indica comprovação parcial da teoria de que: 1. Há erros de rótulo, 2. Há exemplos muito difíceis ou no limiar.

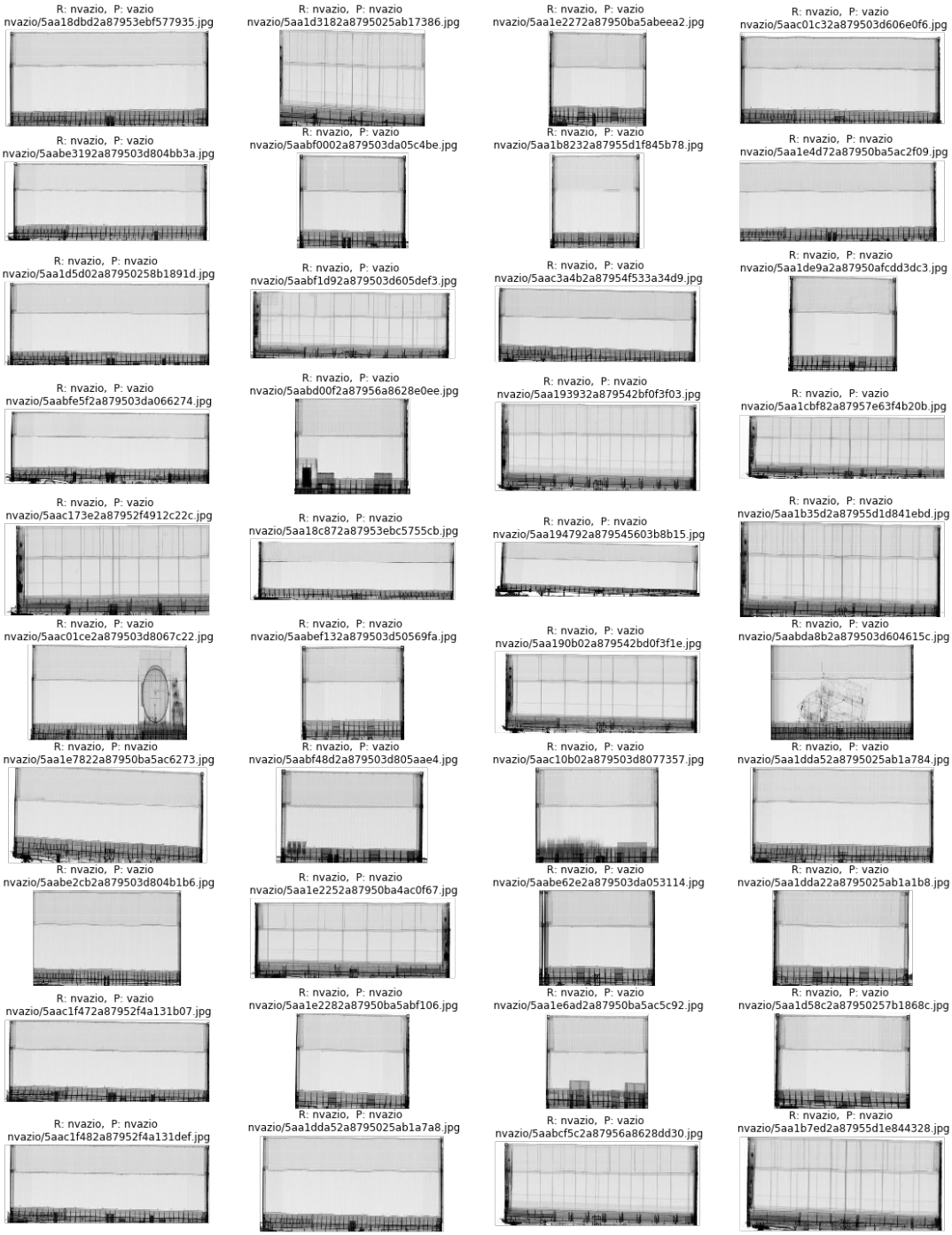
Para lembrar, no início do treinamento o modelo cometia erros como os abaixo, classificando contêineres claramente não vazios como vazios, mas com probabilidade baixa (65% ou menos, sendo que 50% é o limiar para classificação na outra classe).



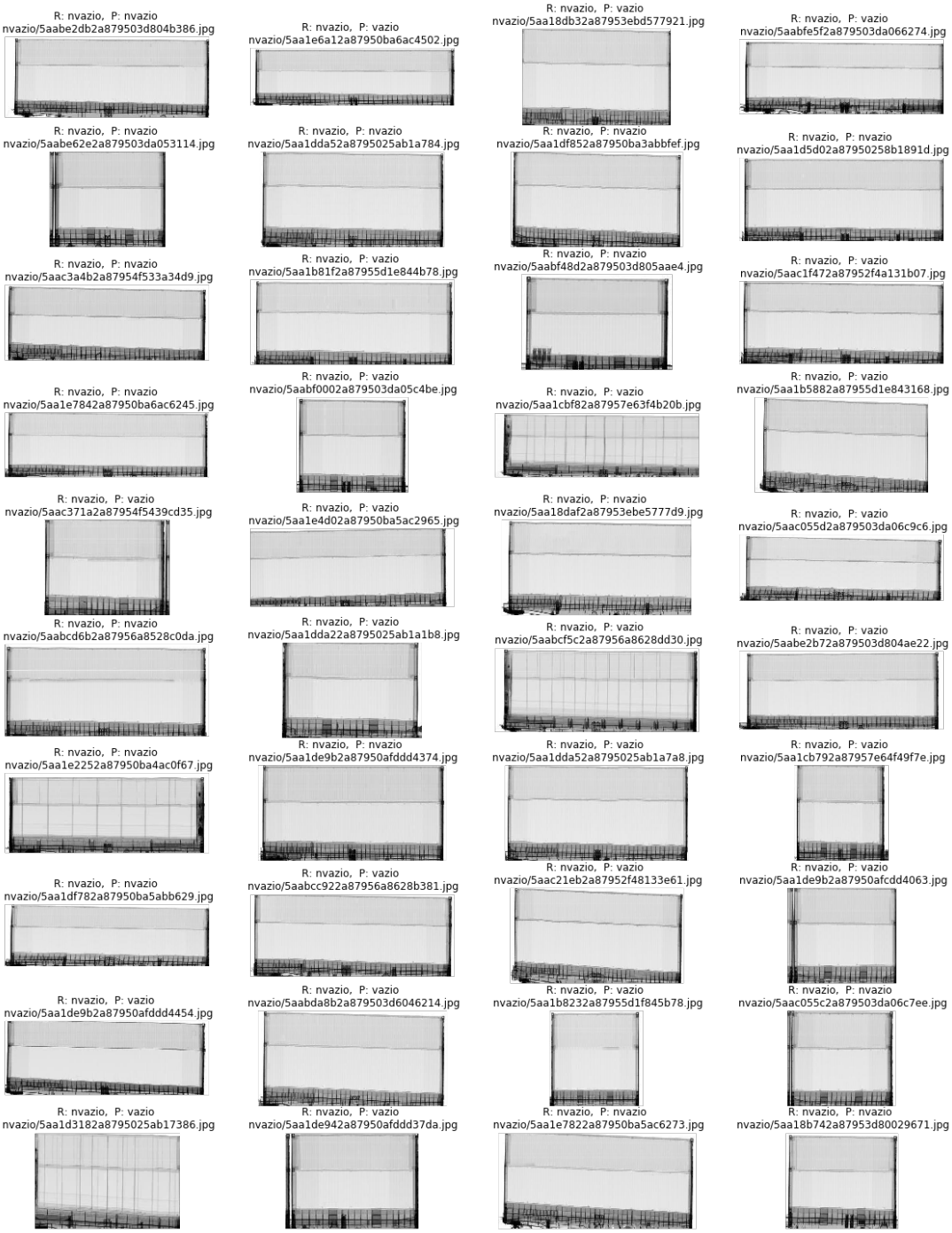
Já os últimos modelos estão cometendo os erros mostrados abaixo. Note-se que o SVM comete erros "bobos". Já os erros de rede neural, especialmente os

com recall mais alto, quase em 100% das vezes fica difícil de saber se é erro de rótulo, pois a visualização também indica um contêiner sem carga.

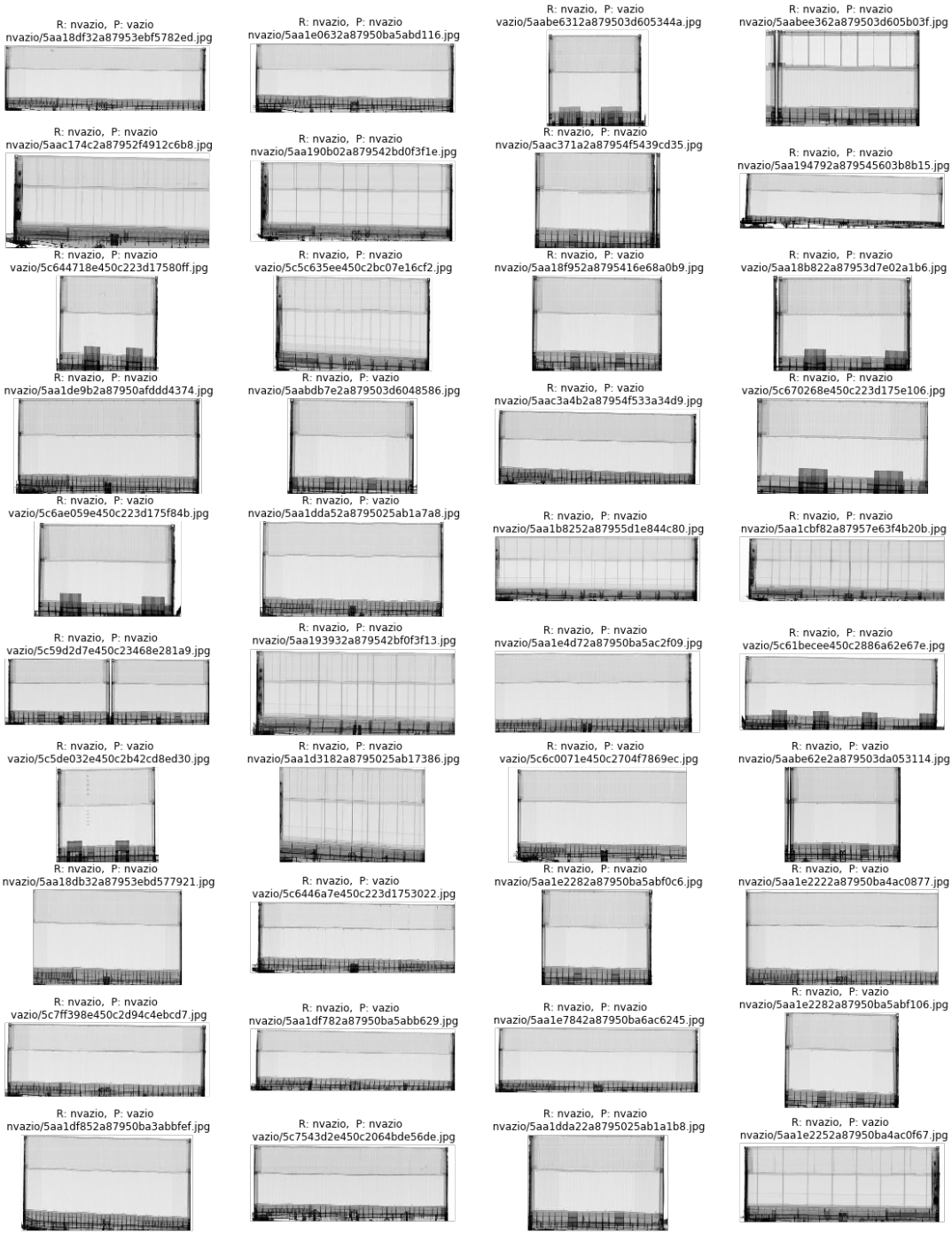
- Erros SVM



- Erros Rede Neural Simples

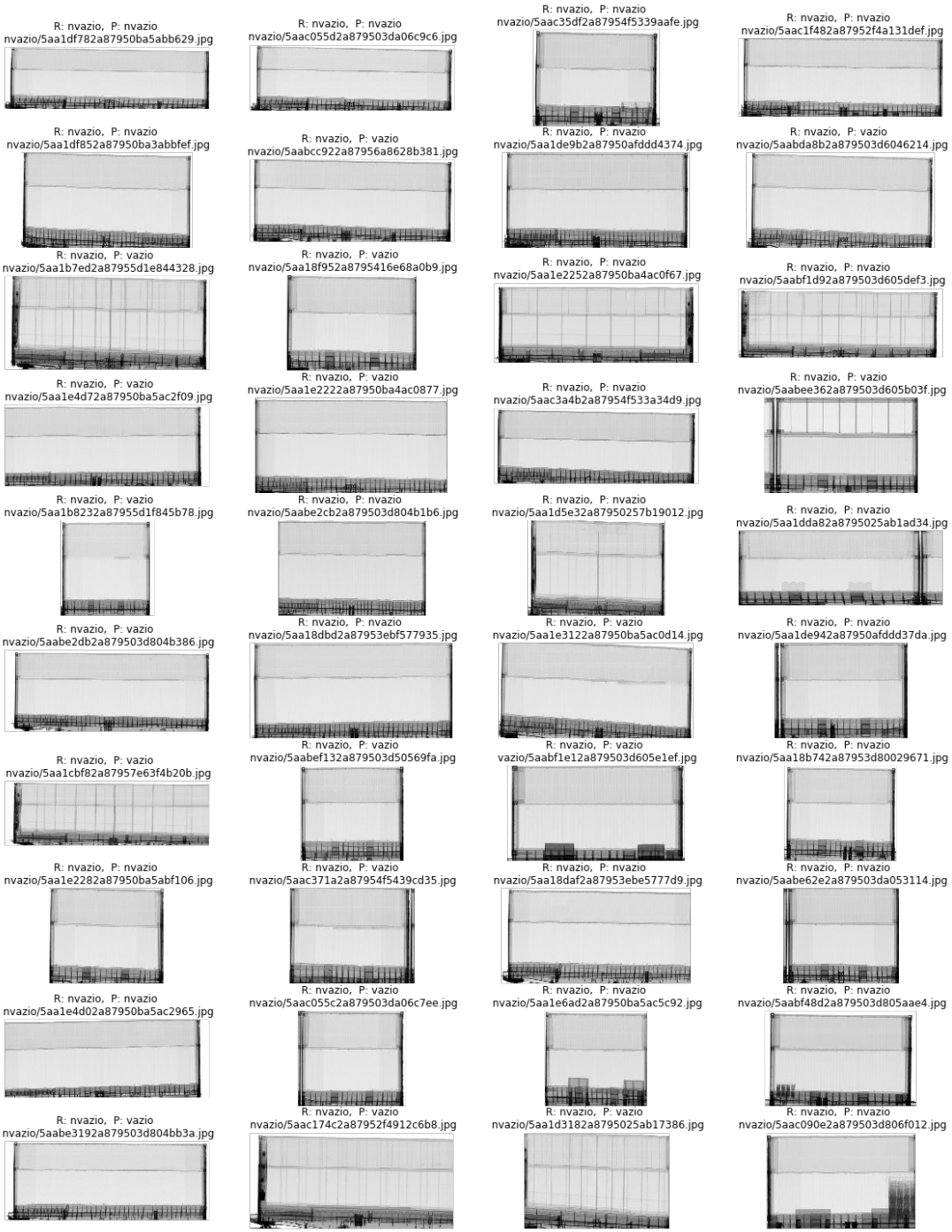


- Erros DenseNet Transfer Learning



- Erros DenseNet Transfer Learning com class weights Erros 3a

- Erros Siamesa



Desempenho (em tempo e consumo de memória)

Devido à grande dimensionalidade do problema, a solução SVM ocupa bastante memória e é a que tem maior tempo de execução (em computador que possui GPU, provavelmente sem GPUs as redes neurais teriam desempenho muito inferior) Em seguida a rede DenseNet pré treinada é a que tem maior uso de

memória. A rede neural simples possui o menor uso de memória de todas e maior rapidez. A rede siamesa utiliza memória intermediária mas se mostrou mais lenta que a rede imagenet, pois precisa extrair features de duas imagens para depois fazer comparações.

Tempos para carregar do disco, redimensionar e fazer predicões em 2317 imagens:

```
Nome da técnica             tempo           Consumo de
memória          GPU(1)
Rede DenseNet121            2min 35s
Alto                    Não
SVM                         1min 37s
Alto                    Não
Rede DenseNet121            25.2 s
Alto                    Sim
Rede neural simples 01b3    22.2 s
Alto                    Não
Rede siamesa(2)             15.4 s
Médio                   Sim
Rede neural simples 01b3    9.15 s              Médio-
baixo             Sim
Tratar 1024 features(3)      43 ms
Baixo                   Não
Tratar 1024 features(3)      126 ms
Baixo                   Sim
Tratar 128 features(3)      <10 ms              Muito
baixo             Sim
```

1. GPU não é estritamente necessária para predição de redes neurais, mas em uma predição de rede neural, especialmente uma rede convolucional, o tempo é muito maior sem utilização de GPU. Para este teste foi utilizada CPU Intel i5 com 2.30ghz e 8 cores, cache L1 32K, L1 256K e L3 8192K e NUMA mode. A GPU utilizada foi uma GTX1050ti 4GB, cuda 10.1. Memória do Sistema de 8GB.
2. A rede siamesa precisa sempre passar duas imagens, por isso o tempo é maior que 01b3. Para melhorar a precisão, poderiam ser comparadas vários pares de imagens, e os tempos serão somados
3. Desde o começo do projeto uma das idéias que saem do padrão de simplesmente treinar um classificador do iníco ao fim é reutilizar os aprendizados de uma etapa em outra. Assim, caso as imagens estejam em

uma base centralizada, uma boa prática que quase nunca é vista nos *papers* ou tutoriais seria fazer pré extração de *features* em batch e utilizar, posteriormente, estas *features* ao invés de carregar a imagem original. É uma redução entre 2.400 a 20.000 vezes (de uma imagem SVGA para 1.024 ou 128 floats por exemplo). Assim, economiza-se disco, I/O, processamento, memória, energia elétrica e a redução drástica do tempo de processamento possibilita inclusive fazer tarefas muito mais complexas, como uma busca de similaridade em todo o banco de dados ou agrupamento (clusterização) das imagens.

Desenvolvido na RFB dentro do escopo do Sistema AJNA

Ivan da Silva Brasílico

Código Fonte no GitHub

- Apresentado como Capstone Project no curso de Engenheiro de Machine Learning, Udacity.