Universidade Federal de Pernambuco
Centro de Informática - CIn
Graduação em Ciência da Computação

Reconhecimento de ambientes em fotos do Instagram com *deep learning*

Rubens Lopes de Farias Silva

Trabalho de Graduação

Recife, Dezembro de 2016

Universidade Federal de Pernambuco
Centro de Informática - CIn

Graduação em Ciência da Computação

Reconhecimento de ambientes em fotos do Instagram com *deep learning*

Trabalho apresentado ao Programa de Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Aluno: Rubens Lopes de Farias Silva
(rlfs@cin.ufpe.br)

Orientador: Germano Crispim Vasconcelos
(gcv@cin.ufpe.br)

Recife, Dezembro de 2016

“Não é sobre chegar no topo do mundo
saber que venceu, é sobre escalar e sentir
que caminho te fortaleceu.”
Ana Vilela - Trem Bala

Agradecimentos

Primeiramente, eu gostaria de agradecer a todos que me apoiaram na difícil decisão de sair do interior de Pernambuco (Caruaru) para cursar o bacharelado em Ciência da Computação no Centro de Informática.

Agradeço à minha mãe (Dilma), ao meu pai (Edson) e ao meu irmão (Robson) pelo grande apoio e suporte dados ao longo de todo o trajeto, desde do início do curso até agora. Pelas palavras de incentivo quando achava que o curso estava muito difícil ou quando as dificuldades de morar em outra cidade vinham à tona.

Agradeço à Joseane por sempre acreditar no meu potencial e acreditar que sou capaz de chegar onde eu quero chegar. Agradeço à Carla dos Santos pela ajuda no processo de desenvolvimento deste trabalho. Agradeço à Danielle Souza por me mostrar onde o caminho da educação poderia me levar.

Agradeço à UFPE e especialmente ao Centro de Informática (CIn) pela excelente formação que tive. Pela oportunidade de aprender tudo aquilo que será essencial para continuar progredindo na minha carreira. Pelos projetos que pareciam impossíveis, mas que eram extremamente recompensadores quando estavam concluídos. Pelos monitores que mostraram que todos estávamos juntos na luta e que ajudando uns aos outros conseguiríamos ir muito mais longe. Agradeço também ao o DAE pelo apoio de moradia na Casa do Estudante, que foi muito importante para que eu pudesse focar mais nas graduação.

Agradeço aos meus amigos que tornaram a minha graduação incrível. Pelos sorrisos, brincadeiras, noites viradas, projetos gigantes e caronas, momentos que sentirei saudades com certeza. Especialmente a Bruno Assis, Dyego Felipe, João Pedro, e Pedro Torres que sempre nos apoiamos durante todo o curso.

Por fim, agradeço a todos os que foram fundamentais na transformação da minha graduação em não apenas uma graduação técnica, mas uma formação para a vida.

Resumo

O uso de sistemas de recomendação vem tornando-se importante no meio corporativo. Desse modo, esses sistemas precisam criar um perfil de usuário e ir além de apenas uma lista de produtos comprados. As informações adicionais necessárias para construir o perfil do usuário geralmente estão em redes sociais. Porém, algumas delas, como o Instagram, têm boa parte do seu conteúdo apresentado em imagens, que não são fontes de informações textuais, dificultando assim o processamento dos sistemas de recomendação tradicionais. Portanto, este trabalho tem como objetivo criar uma solução para a classificação de imagens das fotos do Instagram, com o foco de identificar quais ambientes elas foram tiradas. A solução foi construída com a rede neural artificial *Deep Learning,* devido aos recentes bons resultados obtidos em classificação de imagem. Para exemplificar, foram escolhidos três ambientes: academia, casa noturna e praia.

Abstract

The usage of recommendation systems has become important in the business environment. In this way, these systems need to create an user profile and go beyond just a list of purchased products. The additional information needed to build the user profile is usually on social networks. However, some social networks, such as Instagram, has most of their content presented in images, which are not textual information, making it difficult to process by traditional recommendation systems. Therefore, the goal of this paper is to create a solution for image classification of Instagram photos, with the focus of identifying the environments where these photos were taken. The solution was built with the deep learning artificial neural network, which has obtained recent good results in image classification. To exemplify, three environments were chosen: gym, nightclub and beach.

Abreviações

Mineração de Dados (MD)

Sistema de Recomendação (SR)

Deep Learning (DL)

Classificação de Imagem (CI)

Redes Neurais Convolucional (ConvNet)

Visão Computacional (VC)

Aprendizagem de Máquina (AM)

Redes Neurais Artificiais (RNA)

Multilayers Perceptron (MLP)

Sumário

1 INTRODUÇÃO

Devido à grande quantidade de informação gerada pelos usuários através do uso dos computadores, empresas que visam usar esses dados como uma vantagem competitiva precisam criar meios eficientes de interpretar dados dos usuários. Segundo T. Y. Lin (1997), grandes bases de dados contêm o potencial de uma mina de ouro de informações. Logo surgiu a área de Mineração de Dados.

Mineração de Dados, do inglês *Data Mining* (DM), é definida por J. Han (2001) como "o processo de descoberta de conhecimento interessante em grandes quantidades de dados armazenados em Bases de Dados". Logo, podemos dizer que DM pode oferecer recursos de identificação de perfil de usuário, para que assim seja possível construir um marketing direcionado para cada perfil de usuário ou um sistema de recomendação (SR) mais eficiente, argumento suportado por Pine (1993), quando escreveu que empresas precisam ir além do velho mundo de produção em massa, produtos padronizados, market homogêneo e produtos para uma vida toda, para um novo mundo onde customização substitui produtos padronizados.

Considerando que boa parte destes dados estão nas Redes Sociais e que as informações tendem a ser de caráter pessoal, o estudo do uso de DM em redes sociais torna-se bem atrativo para servir de base para SR. Sem dúvida, Redes Sociais estão repletas de informações úteis, porém, estas informações podem estar em outros formatos, no qual os SR tradicionais, que focam em texto, podem possuir dificuldades de classificação, como áudio, vídeo ou foto. Esse déficit dos SR se agrava mais ainda quando uma rede social está focada em uma mídia não tradicional como o Instagram[1] que é uma rede social de fotos.

Para vencer essa dificuldade, é preciso agregar ao SR, um sistema de Classificação de Imagem (CI). Esse sistema de CI será responsável por analisar a imagem e retornar para o SR as informações extraídas da imagem. Os Classificadores de imagem geralmente são construídos com técnicas de Aprendizagem de Máquina.

[1] www.instagram.com

Com base nas análise das soluções atuais sobre Aprendizagem de Máquina para Classificação de Imagem, temos que os as Redes Neurais com *Deep Learning,* especialmente as Redes Neurais Convolucional (ConvNet), vem apresentando resultados incríveis, como evidenciados por Ren & Ramanan (2013), quando, que cita os resultados recentes têm mostrado que, moderadamente, *Deep Models* estão obtendo melhor performance que o estado da arte da extração de características em gradiente de histograma na detecção de modelos.

Dessa forma, este trabalho teve como objetivo analisar a aplicação de uma ConvNet na classificação de imagens retiradas do Instagram, visando a identificação do perfil do usuário a partir de suas fotos. Em caráter de exemplificação, foi escolhido a análise de ambiente no qual as fotos foram tiradas, levando em conta que os locais frequentados pelo usuário podem auxiliar na identificação do padrão de consumo do mesmo. No caso, foi escolhido os ambientes de Academia, Casa Noturna e Praia, levando em conta a quantidade de fotos postadas e a relevância dos mesmos.

A estrutura deste documento está dividida capítulos. O Capítulo 2 apresenta uma breve fundamentação sobre a importância da mineração de dados para os Sistemas de Recomendação, além de uma breve explicação de como as Redes Neurais Deep Learning e ConvNet funcionam, particularmente a rede TensorFlow[2], escolhida para ser aplicada neste trabalho. Já a justificativas das escolhas feitas no projeto e implementação estão relatadas no Capítulo 3, assim como os resultados dos experimento e os das melhorias aplicadas estão no capítulo 4. Por fim, capítulo 5, onde a conclusão e indicação de trabalhos futuros são relatadas.

[2] www.tensorflow.com

2 CONTEXTO

O objetivo deste capítulo é explicar e fundamentar alguns conceitos em que este trabalho está inserido, para que o leitor esteja mais contextualizado sobre o estado da arte, das técnicas e das ferramentas usadas, além de justificar algumas escolhas feitas do trabalho.

2.1 Mineração de dados

Diariamente, a quantidade de dados produzida e armazenada está muito além da capacidade humana de análise sem auxílio de ferramentas computacionais. Pensando nisso, no final dos 80 surgiu uma técnica, chamada Mineração de dados (*Data mining*), com o objetivo ajudar a extrair conhecimento de grandes bases de dados.

Com o passar do tempo, os objetivos da mineração de dados (DM) se tornaram mais abrangentes e fundamentais para vários ramos, como foi argumentado por S. Amo (2014): ”Atualmente, Data Mining consiste sobretudo na análise dos dados após a extração, buscando-se por exemplo levantar as necessidades reais e hipotéticas de cada cliente para realizar campanhas de marketing”. Portanto, é importante para as empresas o uso da DM para extrair conhecimentos de seus clientes, fazendo com que seja capaz de interpretar seus objetivos expectativas e desejos (Braga, 2005), logo as empresas podem usar DM como uma ferramenta para criar uma vantagem competitiva na tentativa de atrair mais usuários e fazer um marketing mais inteligente e direcionado a cada cliente.

2.2 Sistema de recomendação

Muitas ferramentas foram criadas com base nas vantagens do uso de mineração de dados. Dentre elas, pesquisas começaram a criar sistemas de recomendação que usam o conhecimento extraído dos dados registrados, usando técnicas de DM, sobre um determinado cliente ou clientes que possuam o perfil de consumo e comportamento similar para indicar futuros serviços ou produtos que sejam mais provavelmente voltados para cada cliente, aumentando assim o consumo.

Desde então, grandes empresas fazem uso de sistemas de recomendação para conseguir proporcionar um tratamento especial para cada tipo de cliente. Por exemplo, a Netflix[3] leva em consideração os filmes já assistidos pelo usuário para indicar filmes similares ou construir um perfil do usuário para melhor indicar filmes baseados em pessoas com perfil similar. Outro exemplo é a Amazon[4], que na compra ou busca de algum produto, constrói uma lista de preferências para o usuário, que será usada futuramente nas indicações de produtos e campanhas publicitárias direcionadas a esse cliente.

Os SR atuais procuram não apenas conseguir indicar produtos similares aos já adquiridos, mas construir sistemas que criem perfis de usuários com seus desejos de comprar e informações de contexto que possam ser relevantes na escolha de um produto. Esse argumento é suportado por C. K. Pereira (2014):

> *O objetivo principal é a identificação de características do perfil e do contexto do usuário a partir de informações geradas, espontaneamente, através do uso das redes sociais, para auxiliar no processo de seleção e recomendação de recursos adequados a esse perfil.*
> *(C. K. Pereira, 2014)*

Um exemplo hipotético, bem prático, é: supondo que uma empresa deseja escolher alguns clientes-alvos para realizar uma campanha publicitária visando promover algum tipo de produto esportivo de um determinado time. Seria muito mais interessante se a empresa soubesse previamente quais dos possíveis clientes-alvos torcem para esse mesmo time. Nesse caso, as informações não necessariamente estariam atreladas à compras antigas efetuadas pelos clientes, mas sim a alguma fonte de dados de cunho pessoal de cada um deles, como uma rede social.

[3] Website: <www.netflix.com>
[4] Website: <www.amazon.com>

2.3 Rede Social

As redes sociais digitais se tornaram uma das ferramentas de grande sucesso criada com a WEB 2.0. As redes sociais são meios de comunicação nos quais os usuários podem publicar informações semi-públicas sobre o seu dia a dia, ler publicações de outras pessoas, além de interagir por meio de comentários.

De acordo com o site Statistica[5], cerca de 1.96 bilhões de pessoas no mundo possuem uma conta de acesso a algum tipo de rede social e a estimativa é de 2.5 bilhões em 2018. Portanto, vemos que as redes sociais tem se tornado cada vez acessadas e difundidas, gerando, assim, para nosso caso, mais dados para os sistemas de mineração de dados.

Vale salientar que as redes sociais sociais são responsáveis por boa parte das informações pessoais que são postadas na internet, à frente de blogs[6], canais de youtube ou similares. Desse modo, os sistemas de recomendação conseguem, a partir de redes sociais, traçar um perfil próximo ao do usuário.

Entre as redes sociais mais famosas atualmente está o Instagram, uma rede social focada em fotos, onde os usuários podem postar fotos pessoais, seguir amigos, curtir e comentar nas fotos que os amigos postaram. Segundo o Statistica[7], o Instagram obteve cerca de 500 milhões de usuário ativos em junho de 2016, além de que, segundo o Quora[8], em 2014 o Instagram recebia em média 60 milhões de publicações diárias. Logo, vemos que o Instagram é uma rede social amplamente usada e deve ser levada em consideração pelos sistemas de recomendação.

Porém, os sistemas de recomendação possuem dificuldades de trabalhar com o Instagram, o que ocorre devido ao fato de ser uma rede social de fotos, o que é um problema para os SR tradicionais, pois eles têm facilidade em trabalhar com texto, mas não com fotos. Dessa forma, boa parte da informação seria descartada

[5] Disponível em <https://www.statista.com/statistics/273476/percentage-of-us-population-with-a-social-network-profile/> acesso Novembro, 2016

[6] Website que facilita a atualização rápida do seu conteúdo.

[7] Disponível em <https://www.statista.com/statistics/253577/number-of-monthly-active-instagram-users/> acesso Novembro, 2016

[8] Disponível em <https://www.quora.com/How-many-photos-are-being-uploaded-on-Instagram-daily> acesso Novembro, 2016

se o SR apenas considerar os títulos e comentários das fotos, sem considerar o conteúdo das mesmas. Uma funcionalidade do Instagram é o uso de *hashtags* nos comentários, facilitando sua classificação. porém, muitas vezes as hashtags usadas não condizem perfeitamente com o conteúdo da imagem.

Como solução, os sistemas de SR precisam de uma ferramenta de classificação de imagem para conseguir obter informação também da mesma.

2.4 Visão Computacional

A visão computacional é o conjunto de etapas que processam e interpretam informações contidas em imagens (J. Felix, 2007) com o objetivo de identificar elementos contidos nelas e, a partir desses elementos, extrair algumas informações para um futura classificação. Isso pode parecer muito simples para o ser humano, mas para o computador, uma imagem é apenas um conjunto de Pixels[9] com intensidade de cores.

Um sistema de visão computacional pode ser dividido em várias etapas, que levam desde a obtenção da imagem até a sua classificação. Estas etapas estão descritas na figura 2.4.

Figura 2.4: Fluxograma de um sistema de visão computacional padrão (J. Felix, 2007)

Cada parte desse bloco representa uma etapa no processo de visão computacional. Para melhor entendimento, cada parte do fluxograma será descrito a seguir (J. Felix, 2007):

- **Aquisição de imagens**: Nesta etapa, são tratadas as condições e como a imagem será capturada. A obtenção da imagem pode ocorrer através de câmeras fotográficas, *scanners,* máquinas de raio X, etc. No caso de imagens provenientes de câmeras fotográficas, por exemplo, ocorre a preocupação com a iluminação pré-foto, a escolha da câmera e da lente.
- **Pré-processamento**: a finalidade do pré-processamento é ajustar características das imagens que possam ser relevantes para as próximas

[9] "Um pixel é o menor ponto que forma uma imagem digital." - Wikipedia

etapas do processo. Nessa etapa, são aplicados filtros de correção de iluminação pós-foto, além de ajustes de frequência de pixels e ajustes de cores. Um exemplo é se a imagem deve estar em tons de cinza para a próxima etapa.

- **Segmentação**: a etapa de segmentação é uma das mais importantes do processo de visão computacional, pois o desempenho do sistema está fortemente ligado aos seus resultados . Nela, o sistema procura entender aonde, na imagem, estão as bordas e texturas dos objetos contidos. Esse processo é geralmente baseado na descontinuidade e similaridade de blocos de pixels vizinhos.
- **Extração de atributos**: o objetivo dessa etapa é conseguir agrupar segmentos da etapa anterior com o intuito de criar objetos mais completos que estejam bem definidos em relação a bordas e estrutura interna.
- **Reconhecimento e interpretação**: essa é a etapa final, no qual o sistema tenta identificar os objetivos resultantes da etapa anterior, considerando bordas e texturas. Nesta etapa, o sistema deve extrair padrões e características do objetos para que o mesmo consiga ser identificado.

Na maioria das vezes, a última etapa, também conhecida como classificação de imagem (CI), tem dificuldades de encontrar um padrão matemático de cada objeto que será alvo do sistema de visão computacional. Então o uso de sistemas inteligentes, especialmente aprendizagem de máquina, são bem apropriados nessa última etapa do processo.

2.5 Aprendizagem de máquina

O termo aprendizagem de máquina possui muitas definições. Uma definição bastante usada é dada por Weiss e Kulikowski (1991):

> *"Um sistema de aprendizado [supervisionado] é um programa de computador que toma decisões baseadas na experiência contida em exemplos solucionados com sucesso"*

Diferentemente dos sistemas inteligentes comuns, onde as decisões são tomadas com auxílio de heurísticas matemático ou de conhecimento aprofundado do problema, os sistema de aprendizagem de máquina (AM) procura conseguir adquirir conhecimento sobre determinado assunto baseado em exemplos prévios devidamente registrados. Podemos resumir o conceito de AM como um sistema que recebe valores de entrada e retorna um valor de saída baseado no conhecimento proveniente de uma base de dados.

Um sistema de AM possui duas etapas principais: aprendizagem e produção. Na etapa de aprendizagem, o sistema utiliza exemplos da base de dados para aprender sobre como classificar cada entrada ajustando parâmetros internos, com o intuito de conseguir obter o resultado correto quando um novo padrão de entrada for apresentado. Já na produção, o sistema será usado para prever qual valor melhor valor de saída se encaixa para aqueles valores de entrada apresentados, sem ajustes de aprendizado.

Um das categorias mais comuns e utilizadas sobre sistemas de AM para a etapa de aprendizagem é a de aprendizagem supervisionada.

2.5.1 Aprendizagem supervisionada

Nessa categoria de aprendizagem, os registros da base de dados que serão usados pelo sistema para o treinamento já foram previamente classificada. Nesse caso, o sistema no processo de aprendizagem já possui qual seria o resultado esperado para o determinado padrão de entrada. A partir disso, o sistema tenta se adaptar internamente para minimizar a quantidade de erros até que ele se adeque a um parâmetro aceitável.

Para melhor compreensão sobre aprendizagem de máquina supervisionada, é importante conhecermos alguns termos, que são descritos por Kuncheva (2004):

- Classe: também conhecido como rótulo, é uma valor que determina sobre qual grupo a entrada deve pertencer. Como exemplo, um objeto da imagem pode ser classificado/rotulado como uma bola.
- Atributos: também conhecido como padrão de entrada, são os valores que serão usados pelo sistema para prever a que classe ele pertence. Como exemplo, uma silhueta, cor do objeto.

Na situação ideal, se dois objetos possuem atributos similares, eles deveriam possuir a mesma classe.

2.5.2 Rede Neural Artificial

As redes neurais artificiais (RNA) foram criadas nos anos 40 e ainda hoje servem de base para muitos sistemas inteligentes avançados. A RNA é um sistema de AM inspirado nos sistemas nervosos dos seres vivos pela capacidade de trabalhar com inúmeras tarefas, como visão, audição, fala, etc. (Haykin, 2008). Podemos complementar essa semelhança levando em conta que as RNAs adquirem conhecimentos através de aprendizado e esse conhecimento é armazenados entre conexão entre neurônios (Kuncheva, 2004).

2.5.2.1 Modelo McCulloch e Pitts

O primeiro modelo neurônio artificial foi criado por McCulloch e Pitts no ano de 1943 em um artigo chamado "*A logical calculus of the ideas immanent in nervous activity*". Esse modelo ainda hoje serve de base para muito dos sistemas de RNA. A figura 2.5.2.1 retrata seus funcionamento e entidades.

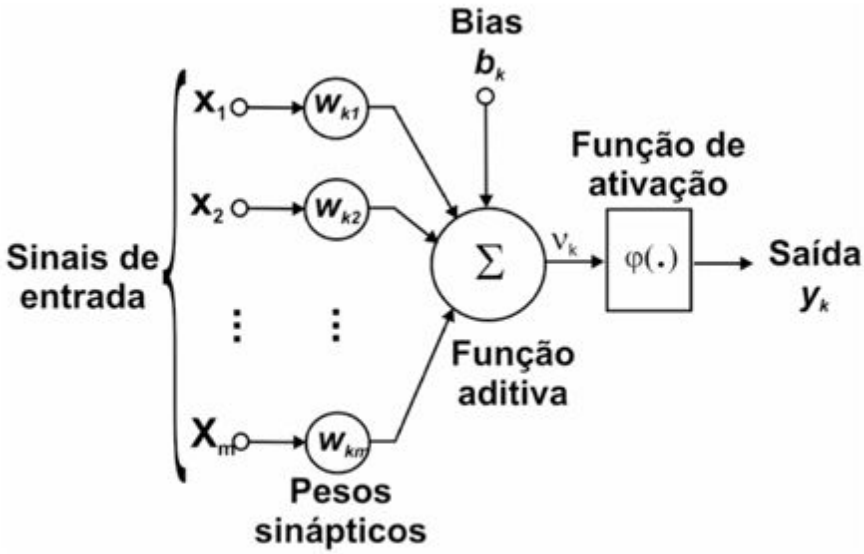

Figura 2.5.2.1 - Modelo de neurônio artificial de McCulloch e Pitts[10]

[10] Disponível em <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0370-44672005000400011> acesso Novembro, 2016.

A lista abaixo possui uma breve explicação sobre cada parte desse modelo:

- **Sinais de entrada**: são os atributos do objeto nos quais a rede se baseia para a identificação.
- **Pesos sinápticos**: são os pesos/relevância dados a cada entrada, os quais irão influenciar na decisão da classificação do objeto.
- **Bias**: um valor de reajuste. Não é necessário para o funcionamento do neurônio.
- **Função aditiva**: uma função de soma de todos os valores.
- **Função de ativação**: é uma função que determina se o valor de saída da função aditiva será o suficiente para ativar o neurônio. Geralmente, é usado um valor de limiar (*threshold*) para essa decisão.
- **Saída**: um valor binário que I representa se o neurônio foi ativado (1) o não (0).

Para melhor entendimento, vamos a um exemplo de funcionamento do fluxo do modelo: Dado um valor de entrada {x1, x2, … , xm}, multiplicamos cada entrada pelo seu respectivo peso sinápticos {wk1, wk2, … ,wkm}. Então, a função aditiva faz um somatório de todos os valores mais o valor do bias (x1*wk1 + x2*wk2 + ... + xm*wkm + bias) e passa esse valor à função de ativação. Caso esse valor seja maior que um limiar φ, a saída recebe valor 1, caso contrário, valor 0.

Esse modelo se tornou bastante usado e referenciado. porém, ele apresenta uma dificuldade: No modelo de McCulloch e Pitts, o usuário precisaria conhecer os valores dos pesos sinápticos para cada entrada. Logo, esse processo de descoberta dos pesos se tornou muito árduo para problemas grandes.

2.5.2.2 Perceptron

Foi então que, em 1957, Rosenblat desenvolveu o perceptron. Essa RNA é capaz de usar uma base de dados para aprender, assim como os modelos de AM. O perceptron usava a base de dados para ajustar seus pesos sinápticos a partir de um algoritmo de aprendizado, baseando-se na saída que desejada para cada padrão de entrada. Dessa forma, toda vez que o perceptron errava a predição, o

algoritmo de aprendizagem procurava ajustar os pesos para que o sistema aprendesse quais os pesos mais próximos do ideal.

A RNA perceptron é usada até hoje. porém, uma das suas limitações é que ela só consegue resolver problemas linearmente separáveis. Entende-se como problemas linearmente separáveis, um conjunto de problemas em que podemos passar uma reta dividindo a base de dados nas classes do problema.

2.5.2.3 Multilayers Perceptron (MLP)

Para solucionar os problemas não linearmente separáveis, Rumelhart, Hinton e Williams publicaram em 1986 o artigo chamado “learning internal representation by error propagation” um modelo de multicamadas do perceptron (*multilayer perceptron)*. O que, resumidamente, é um conjunto de perceptrons organizado de camadas. Esse modelo apresentou muitos bons resultados e é fortemente usado em pesquisas e por indústrias/empresas por conta de seu bom poder de abstração dos dados.

O modelo MLP possui três tipos de camadas: entrada, onde o sistema irá receber os dados; escondida, onde ele irá receber como entrada a saída da camada anterior e sua saída será usada na próxima camada; saída, que irá receber como entrada a saída da última camada escondida e sua saída será a saída de todo o sistema.

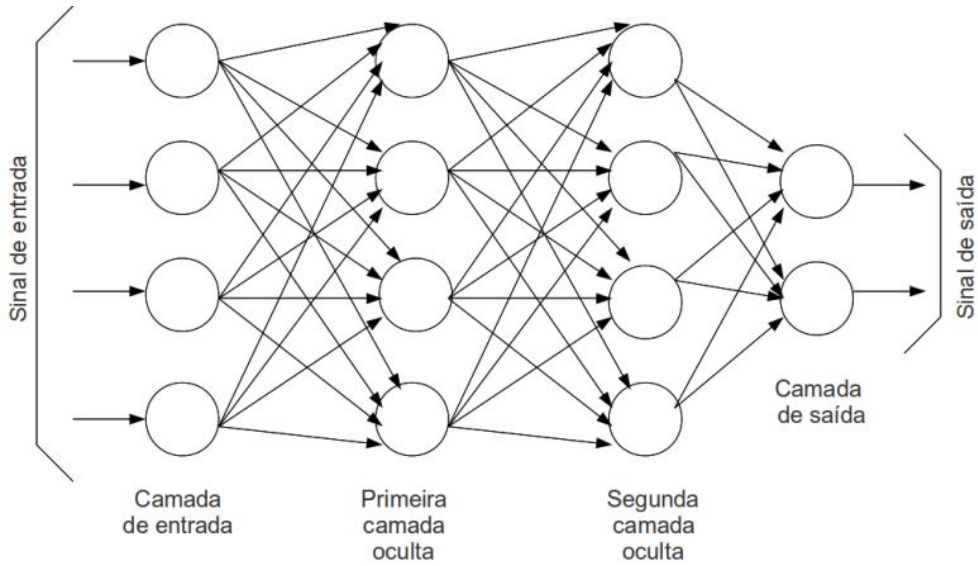

Figura 2.5.2.3 - Modelo *Multilayer Perceptron* (MLP)[11]

[11] Disponível em <https://alexandrevolpi.wordpress.com/tag/dino/> acesso Novembro, 2016

Na figura 2.5.2.3 podemos observar que cada camada da MLP pode possuir vários neurônios, assim como várias camadas escondidas, também que o fluxo é progressivo, o que quer dizer que o dado sempre vai da camada da esquerda para a direita. Outros pontos relevantes foram levantados por Kuncheva (2004):

- O objetivo da camada de entrada é apenas receber os dados, então seus pesos sinápticos são todos 1;
- Os neurônios da mesma camada não se comunicam;
- As camadas escondidas podem possuir números de neurônios diferentes, porém, possuem a mesma função de ativação.

Uma outra adaptação que a MLP precisaria fazer em relação ao perceptron é o algoritmo de aprendizagem. O algoritmo de aprendizagem mais comum usado na MLP é o *backpropagation*. Em poucas palavras, o funcionamento do *backpropagation* é feito da direita para a esquerda atualizando os pesos de cada camada. Essa atualização é feita por um cálculo estipulado do erro de cada camada, destacando que apenas a última camada conhece o erro real, pois apenas ela tem como saída a classe e pode ser comparada com a saída desejada (modelo supervisionado).

2.6 Deep Learning (DL)

Apesar da *deep learning* ser um sistema de AM, por causa de sua importância neste trabalho, foi reservada uma seção no capítulo de contexto apenas para ele.

O *deep learning* pode ser considerado uma evolução da MLP, ainda inspirando-se na biologia. Nesse caso, a inspiração foi do trabalho de Hubel (1968) no qual retratou estudos feitos no córtex da visão de gatos, considerando que o sistema de visão dos animais ainda é o sistema mais poderoso visão existente (Serre, 2007). Nesse caso do estudo de Hubel (1968), foi descrito que o sistema de visão é dividido em sub-regiões e cada uma delas é responsável por uma parte do

objeto, como bordas, cores, profundidade, etc. e no final pro processo a junção delas seria a visão como um todo (deepleanring.net)[12].

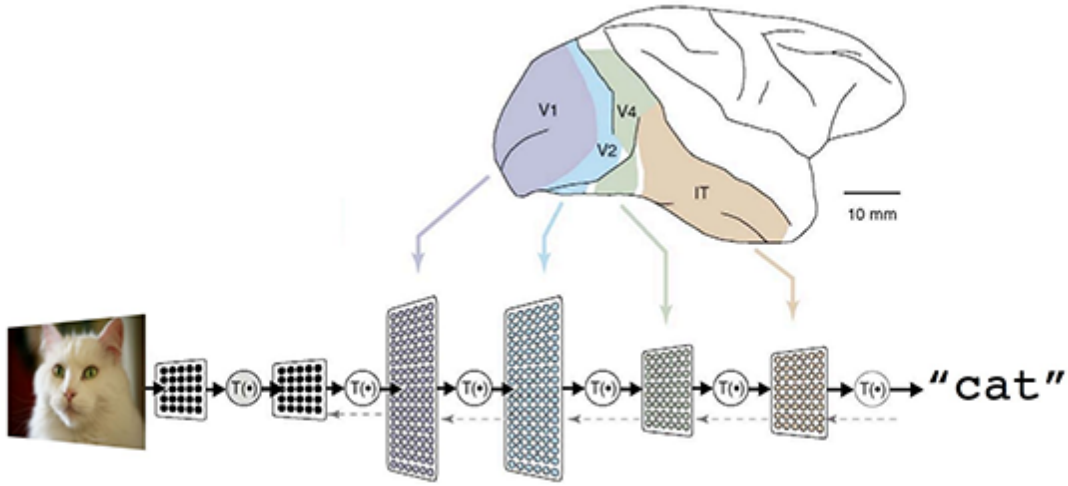

Figura 2.6.1 - Exemplificação de uma rede neural DL[13]

Com esse pensamento, baseado no exemplo da figura 2.6.1, o DL é uma rede neural semelhante a uma MLP, porém com muitas camadas escondidas, onde uma ou um conjunto delas seria responsável por uma parte do processo de classificação.

[12] Disponível em <http://deeplearning.net/tutorial/lenet.html> acesso Novembro, 2016
[13] Disponível em <https://cloud.google.com/blog/big-data/2016/07/understanding-neural-networks-with-tensorflow-playground> acesso Novembro, 2016

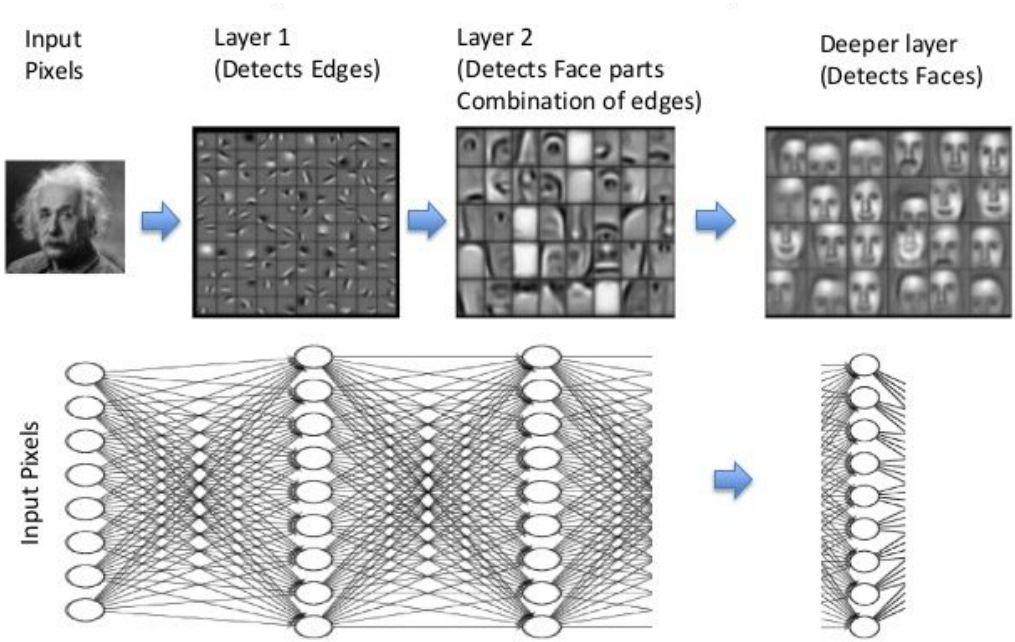

Figura 2.6.2 - Responsabilidade das camadas de uma DL[14]

Como podemos ver na Figura 2.6.2, uma rede neural DL possui várias camadas, onde um conjunto de camadas é responsável por identificar certas características da imagem. Como exemplo um conjunto pode ser responsável pela identificação das bordas, onde o seu resultado pode ser a entrada para outro conjunto de camadas responsável por combinar os estas bordas em uma melhor representação e assim por diante até que no final, todo o conteúdo seria condensado e na camada de saída que faria a identificação. Esse processo é chamado de transferir conhecimento, ou *transfer learning*. Apesar das semelhanças, uma MLP com *backpropagation* possui problemas no treinamento com redes de mais de duas camadas escondidas (Bengio, 2009).

A popularização do DL se deu também pelo avanço na área de hardware para processamento e sua redução de custo, tendo como exemplo GPUs[15], por conta de seu alto custo computacional, além dos avanços na pesquisa de reconhecimento de sinal, que é onde o DL tem os melhores resultados. Por fim, grandes Universidades e empresas passaram a investir em pesquisa neste ramo, como Google, Microsoft, Facebook, MIT, etc. (Deng, 2014).

[14] Disponível em <http://www.slideshare.net/RukshanBatuwita/deep-learning-towards-general-artificial-intelligence> acesso Dezembro, 2016

[15] Graphics Processing Unit

Segundo Deng (2014), o DL tem mostrado bons resultados em diversas aplicações, como visão computacional, reconhecimento de fonética, pesquisa com voz, reconhecimento de discurso convencional, classificação semântica de enunciado, reconhecimento de escrita à mão, processamento de som, reconhecimento visual de objetos, recuperação de informação, e análise de moléculas para auxílio na descoberta em novas drogas.

Sendo o *Deep Learning* apenas um modelo, redes neurais são criadas baseadas no modelo DL. Uma destas redes é a rede neural convolucional, do inglês *convolutional neural networks* (ConvNet), uma rede neural no modelo *deep learning* que foi especialmente desenvolvida para trabalhar com dados de duas dimensões, como fotos e vídeos.

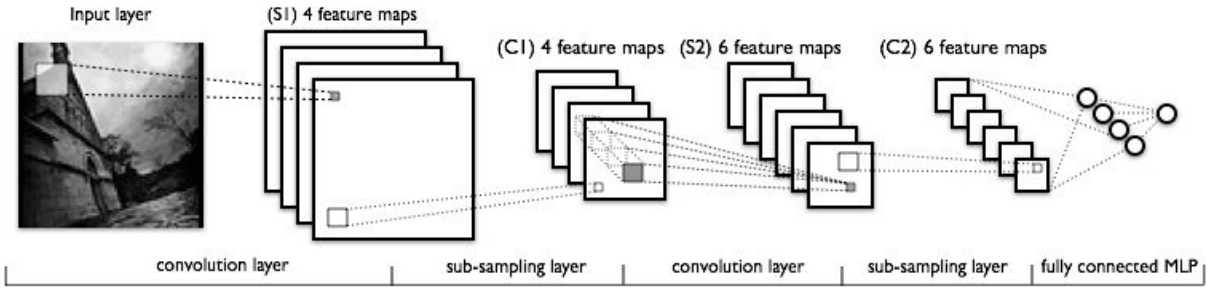

Figura 2.6.3 Estrutura de uma ConvNet[16]

A figura 2.6.3 mostra como uma rede neural DL - no caso ConvNet - funciona. Podemos observar que a saída de cada bloco de característica, que pode ser constituído por várias camadas escondidas, é a entrada do bloco subsequente. Desse modo, cada bloco fica responsável por extrair e mapear uma característica da imagem. Nota-se que a camada final volta a ser semelhante a uma camada MLP comum.

Um dos grandes diferenciais das redes neurais DL é o fato de ela pular muitas etapas no processo de reconhecimento de sinal. No caso da visão computacional, uma rede ConvNet consegue receber muito antes as imagens de entrada comparado ao processo de visão computacional. A entrada pode ser diretamente uma imagem e a ConvNet fica responsável pelo resto do processo de

[16] Disponível em <http://deeplearning.net/tutorial/lenet.html> acesso Novembro, 2016

reconhecimento, demonstrado na figura 2.4 sobre visão computacional, onde o sistema inteligente entrava apenas na etapa final do processo. No caso do uso da ConvNet, o processo pode pular todas as etapas depois da primeira e todo o procedimento excluído será responsabilidade da ConvNet. Ao final, podemos usar a imagem (em algum caso convertida em algum formato especial) diretamente como entrada para a rede ConvNet.

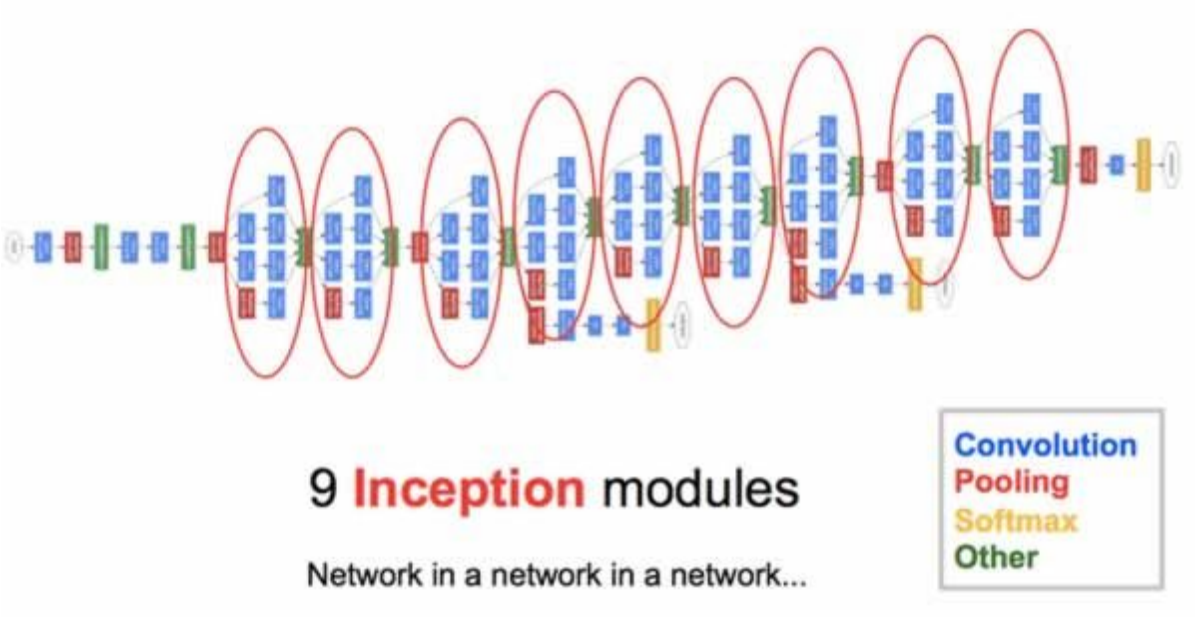

Figura 2.6.4 - Explicação da estrutura ConvNet[17]

Vale ressaltar que na figura 2.6.4 podemos ver que cada extrator de características possui várias camadas com objetivos diferentes, por isso muitas vezes são chamadas de blocos e não apenas de camadas.

Como é de se imaginar, as ConvNet são bastantes complexas e requerem muito tempo para serem treinadas, de modo que a maioria dos tutoriais aconselham fortemente o uso de GPUs para o treino dessas redes. Para minimizar esse problema de tempo no treinamento, pesquisadores disponibilizam algumas ConvNet pré-treinadas para um determinado fim. Geralmente, essas redes possuem algum objetivo amplo, mas o usuário da rede pode decidir apenas re-treinar a última camada da rede que é responsável pela classificação em si, como podemos observar na figura 2.6.5.

[17] Disponível em <https://www.semiwiki.com/forum/content/5930-network-generator-embeds-tensorflow-more-cnns.html> acesso Novembro, 2016

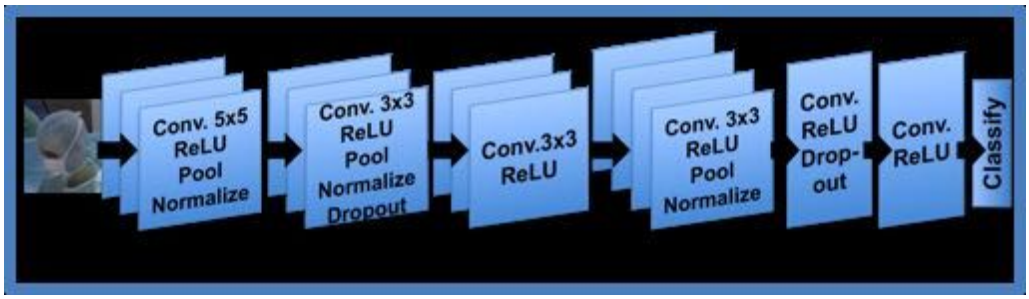

Figura 2.6.5 - Exemplo com camadas nomeadas de uma ConvNet

Uma das ConvNet mais famosas atualmente é a *Tensorflow*. Uma ConvNet desenvolvida pelo Google[18] para a linguagem de programação Python e C++ que propõe reduzir boa parte da dificuldade do uso de um sistema *deep learning*. Apesar de recente (Novembro de 2015), rapidamente se tornou bastante popular. Ela possui muitos tutoriais na internet e tem uma comunidade bastante ativa, porém, ainda contém algumas falhas.

Devido ao grande poder de processamento usado no treinamento da rede, a equipe do *Tensorflow* disponibilizou uma ConvNet chamada *Inception*[19], treinada pela própria equipe da *Tensorflow,* com mais de 1 milhão de imagens de um desafio de classificação de imagem da ImageNet[20], que levou semanas para serem treinadas utilizando vários computadores do Google. O objetivo da rede é conseguir identificar diversos tipos de objetos em uma imagem retorno uma grau de certeza de cada objeto. Entende-se como esse grau de certeza um valor entre 0 e 1 onde no qual 1 representa que o sistema está 100% confiante da sua classificação.

2.7 Base de dados

Outra parte fundamental no uso de sistemas de AM é a preparação da base de dados. Muitas vezes, é preciso preparar os dados para que eles estejam no formato ideal para o sistema de AM, como por exemplo decretar alguma variável ou extrair alguma característica. Porém, como esse trabalho terá como dado apenas imagens, a explicação sobre o tratamento de variáveis será omitida deste trabalho.

Após o tratamento das variáveis, é preciso garantir algumas características sobre os dados:

[18] www.google.com.br
[19] Disponível em: <https://github.com/tensorflow/models/tree/master/inception> acesso Dezembro, 2016
[20] Mais informações: http://image-net.org/

- A primeira é a quantidade de elementos por classe. Como o treinamento de um sistema inteligente se dá a partir dos dados, se uma classe tiver mais exemplos que outra, pode ocorrer de o sistema tornar-se mais tendencioso ao responder à classe com mais registros na base de dados. Portanto, técnicas de samplificação (sampling)[21] são importante para corrigir esse problema;
- Outro ponto importante é a divisão da base de dados em três partes: treinamento, validação e teste. O treinamento será usado para terminar o sistema inteligente; já a validação vai servir para que o sistema tenha uma prévia do seu desempenho; caso este seja ruim, o sistema volta para o treinamento. Esse processo se chama uma época do treinamento; e, por fim, o teste, onde o sistema será avaliado como se ele já estivesse em produção.

Com a base de dados devidamente preparada, o entendimento os algoritmos dos sistemas inteligentes e um bom computador, podemos iniciar o treinamento do sistema inteligente.

[21] Mais informações: https://en.wikipedia.org/wiki/Sampling_(statistics)

3 METODOLOGIA E IMPLANTAÇÃO

O objetivo deste projeto é desenvolver uma aplicação para demonstrar como um sistema de AM pode auxiliar a extração de informação para sistemas de recomendação. Este capítulo contém justificativas sobre algumas das escolhas feitas no desenvolvimento e implementação do projeto.

3.1 Rede social

Como explicado no capítulo anterior, as redes sociais são um meio de comunicação bastante usadas por usuários e boa parte das informações são de caráter pessoal. Então, esse foi o meio escolhido para obtenção das informações dos usuários para o SR. Levando em consideração que o maior déficit dos SR tradicionais é o processamento de mídias não convencionais, foi escolhido o Instagram como a rede social para a obtenção dessas informações, pois é uma rede social focada em fotos.

Boa parte dos SR têm como objetivo auxiliar a área comercial. Portanto, foram escolhidos como informação-alvo alguns temas que são relevantes para essa área. Nesse caso, foi escolhido o reconhecimento de ambientes, pois a partir da informação que o usuário esteve em determinados ambientes, podemos inferir que ele seria mais propenso a voltar a esses ambientes ou a comprar produtos que estão relacionados a eles. Para este projeto, três ambientes foram selecionados por serem bem comuns em fotos do instagram e também serem bem relevantes para a área comercial. Os ambientes selecionados foram: academia, casa noturna e praia.

Vale ressaltar que para uma foto ser classificada como um dos três ambientes, o sistema, além de observar o ambiente em si, também deve considerar elementos da imagem que possam retratar o interesse do usuário no ambiente. Como exemplo, caso o usuário tenha postado uma foto tirada em casa e o sistema tenha encontrado algum tipo de peso de academia na imagem ou a foto foi tirada no espelho mostre o usuário vestindo roupas de malhação, ele deve classificar o ambiente da imagem como academia, pela tendência do usuário a frequentar esse tipo de ambiente.

3.2 Preparação da base de dados

Segundo as políticas de uso do Instagram[22], as fotos postadas nessa rede social estão protegidas por direito autoral, onde o usuário que postou a foto tem o direito sobre ela. Então, apenas sistemas devidamente autorizados conseguem ter acesso a essas fotos através de API[23] (mecanismos que são usados por um sistema para conexão com outros sistemas). Como essa autorização deverá ser solicitada pelo o SR, pois ele seria a solução final, este projeto não utilizou esse método de uso de API para obter as fotos para a base de dados.

Para solucionar o problema de obtenção das fotos, foram usadas duas técnicas:

1. Uma busca era realizada no site do instagram e, por softwares de captura de tela, a foto era selecionada e salva no computador.
2. Uso da ferramenta 4k Stogram[24], que tem o objetivo de fazer o backup de suas fotos, mas também possibilita fazer buscas de fotos públicas do Instagram e salvá-las.

Sem dúvida uma das partes mais complicadas do projeto foi construir a base de dados, pois ela contém apenas fotos do Instagram, que são potencialmente consideradas diferentes das fotos encontradas na internet em geral, e não são facilmente obtidas em fontes de bases de dados comumentes usados em problemas de AM. Logo, foi preciso fazer uma classificação unitária (foto a foto) com a ajuda de um especialista para identificar a classe de cada foto. Esse processo foi extremamente repetitivo e demorado, pois, como era de se esperar, muitas das buscas, mesmo por *hashtags*, retornavam fotos sem conexão com o ambiente, além de uma infinidade de publicidade.

Outro fato de extrema relevância na construção da base de dados foi a variedade de exemplos. No uso real do sistema ele deveria ser capaz de identificar o ambiente em diferentes perspectivas, portanto, seria preciso que a base de dados

[22] Disponível em <https://www.facebook.com/help/instagram/478745558852511> acesso Dezembro, 2016
[23] Mais informações: <https://pt.wikipedia.org/wiki/Interface_de_programação_de_aplicações>
[24] Disponível em <https://www.4kdownload.com/products/product-stogram> acesso Dezembro, 2016

contemplasse cada classe em diversas situações diferentes, como iluminação, foco em objetos, foco em pessoas, etc.

Apesar das fotos para a base de dados serem obtidas sem o uso direto da API, os direitos autorais das fotos foram mantidos. No caso, as fotos contidas neste relatório possuem apenas o objetivo de explicar algum caso especial enfrentado pelo sistema. Mesmo levando em conta que todas as fotos usadas no sistemas foram postadas como restrição pública, qualquer usuário pode visualizar, a base de dados usada neste trabalho não será disponibilizada, porém o modelo da rede neural devidamente treinado (explicado na próxima seção deste capítulo) estará disponível para download, pois não há um método eficiente da obtenção da imagem original a partir do modelo.

A quantidade de fotos para o treinamento variou por experimento, então foi considerado uma porcentagem desses valores para o treinamento, validação e teste. Para a validação foi considerado 10% do valor da base total (valor padrão da ConvNet escolhida) e 30% para o teste, restando assim 60% do valor original da bases para o treinamento. Em todos os experimentos, a quantidade de fotos por classe no treinamento, validação e teste mantiveram a proporção para evitar problemas de samplificação.

3.3 Sistema inteligente

Para a classificação da imagem foi preciso o uso de um sistema inteligente. Foi escolhido um sistema de AM pela dificuldade de extrair regras de classificação de forma manual e a possível base de dados que poderia ser usada para a aprendizagem do sistema.

O sistema de AM escolhido foi o *deep learning*, especificamente uma rede ConvNet, devido aos seus bons resultados na área de classificação de imagem e pelas características das ConvNet no processo de visão computacional, onde o sistema consegue suprir boa parte do processo sem a necessidade de tratamento da imagem e extratores de características adicionais, como explicado no capítulo anterior.

A ConvNet escolhida foi a *Inception V3*, a terceira versão da rede pré-treinada *Inception* disponibilizada pela *Tensorflow*. Essa escolha foi feita pela

disponibilidade do seu uso em Python e pelo objetivo ser similar com o proposto neste projeto, além de seu excelente resultado e tutoriais disponíveis na internet.

Sobre os tutoriais de treinamento e uso da *Inception*, dois tutoriais ganham destaque:

1. O tutorial oficial da *Inception* disponibilizado pela equipe da *Tensorflow.* Nele, o usuário pode aprender como treinar a rede desde do início[25].
2. Um tutorial de como treinar uma rede *Tensorflow* para construir poesia [26]. O mais importante no tutorial são os passos usados para o treinamento e uso. Tais passos podem ser adaptados para o treinamento em outras finalidades.

Ambos os tutoriais oferecem a possibilidade de *freeze* (congelar) os valores atuais da rede para gerar um modelo. Esse modelo possui os valores dos pesos das conexões entre os neurônios do sistema após o treino, logo, ele representa o estado atual da rede e pode ser utilizado em classificações futuras.

O processo do treinamento utilizando a rede *Tensorflow* é dado através de alguns passos. Algumas vezes, alguns desses passos podem ser abstraídos ou alguns outros podem entrar no processo, mas de modo geral, os principais passos são:

- A rede precisa ser retreinada com o uso de um modelo prévio ou um default.
- A etapa do treinamento pode ser finalizada por três motivos:
    - A função de cálculo do erro chegou a 0. A rede acertou todos os casos do conjunto de validação.
    - O valor máximo de épocas, estipulado pelo usuário, foi alcançado.
    - A rede apresenta vários crescimentos na função erro seguidas.
- Após o treinamento, o sistema retorna um modelo.

O fluxo da rede *Tensorflow* para classificação pode partir de um treinamento da rede e seu uso logo em seguida ou a importação de um modelo pré-treinado, como exemplo a *Inception*.

[25] Disponível em <https://github.com/tensorflow/models/tree/master/inception> acesso Dezembro, 2016
[26] Disponível em <https://codelabs.developers.google.com/codelabs/tensorflow-for-poets/#0> acesso Dezembro, 2016

3.4 Métricas de avaliação dos resultados

Para avaliar os resultados do trabalho, foram usadas métricas bastantes comuns em problemas de classificação.

3.4.1 Tabela de confusão

A tabela de confusão é uma tabela criada com os resultados de uma classificação, contém informações sobre o desempenho do sistema na classificação dos testes. Ela é preenchida de acordo com os valores previstos pelo sistema e os valores reais dos dados na etapa de teste.

		Valor Verdadeiro (confirmado por análise)	
		positivos	negativos
Valor Previsto (predito pelo teste)	positivos	**VP** Verdadeiro Positivo	**FP** Falso Positivo
	negativos	**FN** Falso Negativo	**VN** Verdadeiro Negativo

Tabela 3.4.1.1 - Tabela de confusão[27]

A tabela 3.4.1.1 representa o modelo básico de uma tabela de confusão, onde cada elemento será explicado a seguir:

- VP (Verdadeiro Positivo) - Quantidade de entradas que foram previstos como pertencente a classe e realmente os valores pertencem à classe.
- FP (Falso Positivo) - Quantidade de entradas que foram previstos como pertencente a classe, porém os valores reais mostram que a entrada não pertence à classe.
- FN (Falso Negativo) - Quantidade de entradas que foram classificados como não pertencente à classe, porém segundo a base de teste, a entrada deveria ser classificada como positiva

[27] Disponível em <http://crsouza.com/2009/07/13/analise-de-poder-discriminativo-atraves-de-curvas-roc/> acesso Dezembro, 2016

- VN (Verdadeiro Negativo) - Quantidade de entradas que foram classificados como não pertencentes à classe, e realmente a entrada não pertence a classe.

Esses valores conseguem representam o quão bom o sistema está treinando baseado nas respostas dadas pela classe de teste.

Classes reais (gabarito)	Classes atribuídas pelo classificador			
	Classe 1	Classe 2	...	Classe N
Classe 1	n_{11}	n_{12}	...	n_{1N}
Classe 2	n_{21}	n_{21}	...	n_{2N}
. . .	. . .	. . .	. . .	. . .
Classe N	n_{N1}	n_{N2}	...	n_{NN}

Tabela 3.4.3.2 - Tabela de confusão múltiplas classes[28]

Classe ideal (gabarito)	Acuidade na classificação dos pixels (%)		
	Classe fundo	Classe azul	Classe verm
Classe fundo	99,89%	0,09%	0%
Classe azul	0,32%	99,67%	0%
Classe verm	5,08%	5,08%	89,83%

Tabela 3.4.3.3 - Exemplo de tabela de confusão múltiplas classes [27]

Para problemas de múltiplas classes, que é o usado neste trabalho, a tabela de confusão funciona de forma parecida. Explicando um pouco a tabela 3.4.1.2, os valores de linhas são as classes reais do sistema, e as colunas representam qual classe o sistema previu quando estava avaliando as entradas da classe em questão. Em outras palavras, o

[28] Disponível em <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0103-17592007000200010> acesso Dezembro, 2016

valor de “n11” representa quantas vezes o sistema reconheceu como classe 1 as entradas que são da classe 1 como exemplificado na tabela 3.4.1.3.

3.4.2 Métricas de qualidade

Muitas métricas podem ser retiradas da tabela de confusão. As três mais comumente usadas, até pelo seu valor informativo, são as seguintes:

- Precisão - é um valor que representa a quantidade de verdadeiros positivos foram classificados corretamente entre os valores dados como positivos pelo classificador.
- Recall - é um valor que representa a quantidade de verdadeiros positivos foram classificados corretamente entre os valores que deveriam ser classificados como positivos pelo classificador.
- F1 - é uma junção das duas métricas, precisão e métrica. Esse valor representa melhor a equivalência entre os dois resultados.

Um ponto importante sobre a métrica é a soberania em alguma dessas métricas. Por exemplo para um resultado ser considerado bom o valor de precisão e de recall precisam ser altos, consequentemente o valor de F1 alto. Isso é relativamente porque se o sistema se propor a responder tudo como positivo, ele terá uma precisão muito boa, mas seu recall será muito baixo. O oposto acontece caso o sistema rejeite todas as entradas apresentadas.

Cada um desses valores podem ser obtidos com uso de equações matemáticas:

- Precisão = TP / (TP + FP)
- Recall = TP / (TP + FN)
- F1 = (2 x Precisão x Recall) / (Precisão + Recall)

Para tabelas de confusão de múltiplas classes esse valores podem ser calculados considerando os valores de FP, como todos os valores que foram classificados como positivo, porém estão errados. De forma semelhante os FN, onde o sistema não classificou a entrada como a classe em questão, mas deveria classificá-la.

4 EXPERIMENTOS E RESULTADOS

O objetivo deste capítulo é compartilhar experiências, positivas e negativas, no processo de treinamento e experimentos e também mostrar os resultados obtidos da rede e as melhorias realizadas em cada experimento.

4.1 Preparação para experimentos

Como primeira tentativa de solução para o projeto, foi feita uma tentativa de retreinamento de toda a rede *Inception* pois, apesar da proposta final da rede ser bastante similar, os objetos focos de classificação são diferentes. Essa tentativa foi baseada no primeiro tutorial apresentado no capítulo anterior. O objetivo do retreinamento da rede seria fazer aprimoramento para que os extratores de características e afins (primeiras camadas da rede) também fossem retreinados com a tarefa de identificar ambientes e não objetos. Essa tentativa foi frustrada devido ao tempo de treinamento ser extremamente elevado. Nesse caso foi usado um computador com processador i5-4590 CPU 3.30GHz, 16 Gigabytes de memória ram e uma GTX 970 GeForce como GPU com HD SSD. O retreinamento rodou por cerca de uma semana ininterrupta e foi incapaz de chegar perto do seu término. A função *Loss*[29], uma métrica que indica o quanto a rede está acertando em relação à validação, iniciou com o valor 15 e ser reduziu para 7 em um dia, porém, esse valor manteve uma queda muito pequena por muito tempo, até que no final do período de retreino (cerca de uma semana), esse valor estava ainda próximo de 5, onde o ideal é chegar a 0.

A segunda tentativa de solução para o projeto foi o retreino apenas da última camada, responsável pela classificação, como sugerido pelos usuários e pesquisadores de redes ConvNet. Nesse caso, usamos toda a estrutura oferecida para classificação de imagens da *Inception* e apenas retreinamos a rede para classificar os tipos de imagens que queremos. Essa tentativa foi baseada no segundo tutorial também descrito no capítulo anterior. Neste projeto, a rede foi retreinada para identificar apenas as três classes que escolhemos para o projeto. O

[29] Mais informações: https://en.wikipedia.org/wiki/Loss_function

treinamento desta última camada durou apenas alguns minutos, bem diferente da primeira tentativa.

Os experimentos descritos a seguir foram realizados no mesmo computador usado para treinar a rede neural. O objetivo deles foi incrementar e demonstrar quais as escolhas feitas a partir dos resultados da rede nas situações encontradas durante seus testes do desempenho. É importante ter em mente que para os experimentos foram usadas apenas a rede da segunda tentativa de treinamento, pois a primeira não teve resultado em tempo hábil.

Os experimentos foram feitos usando o algoritmo apresentado no Anexo I, um código em *Python* que carrega o modelo treinado e busca por subpastas em uma pasta destino que representa as imagens de teste de cada classe. O fluxo comum dos experimentos é retreinar a rede *Inception* padrão por 200 épocas e depois, usar modelo resultado para classificar a base de teste. No término da classificação, o sistema retorna a taxa de certo de cada classe.

4.2 Experimento 1 - Criação da base manual

Este experimento foi realizado usando a base de dados baseadas na primeira técnica de criação de base de dados. No caso, devido a dificuldade de aquisição das imagens, a base de dados continha apenas 100 imagens em cada classe, onde 70 foram destinadas para o treinamento e validação e 30 para os testes.

Resultados do experimento 1:

	Praia	Casa Noturna	Academia
Precisão	0.9	0.93	0.93
Recall	0.9	0.9	0.96
F1	0.9	0.91	0.94

Tabela 4.2 - Resultado do experimento 1

Podemos observar na tabela 4.2 que mesmo com uma base de dados reduzida, os resultados apresentados pelo sistema já possui bons resultados. Porém como os ambientes podem variar bastantes, é levantado a questão que não

podemos afirmar que o sistema é capaz de generalizar para todos as características dos ambientes com apenas essa base de dados.

4.3 Experimento 2 - Criação da base com o 4k Stogram

O segundo experimento teve o objetivo apresentar ao sistema uma base de dados maior, para que ele se aproxime mais de uma situação real, tanto no treinamento quanto no reconhecimento.

Para este experimento, foi modificado o método de aquisição da base de dados para o segundo método que usa o software *4k Stogram.* Um ponto importante é que a maior parte das imagens adquiridas pelo uso do *4k Stogram* são baixadas corrompidas, porém algumas delas conseguem ser lida pelo o sistema com apenas um aviso.

Para este teste, a base de dados já continha 700 imagens de classe, no qual 490 foram destinadas para o treinamento e validação e 210 foram destinadas para o teste.

Resultados do experimento 2:

	Praia	Casa Noturna	Academia
Precisão	0.99	0.93	0.97
Recall	0.98	0.97	0.94
F1	0.99	0.95	0.95

Tabela 4.3 - Resultado do experimento 2

Os resultados extraídos da tabela de confusão (tabela 4.3) mostrou resultados semelhantes do experimentos 1. Esse experimento mostrou que com mais exemplos, tanto no treinamento quanto na teste, o sistema obteve uma melhora, porém o mais importante é um aumento na capacidade de reconhecer ambientes por mais aspectos.

4.4 Experimento 3 - Pré-classificação de novas imagens

Com o resultado do experimento 2, vimos a expansão da base de dados trouxe um resultado mais concreto e confiante. Então foi estudado uma forma de obtenção de mais imagens classificadas para a base de dados de forma mais automática. As imagens adquiridas pelo *4k Stogram* não possuem uma classificação prévia, então para expandir a base de dados foi usado o modelo (ConvNet treinada) gerado pelo o experimento 2 como um classificador prévio das novas imagens. Desta forma, as imagens possuiriam uma pré-classificação e o trabalho da classificação manual se reduziria bastante.

Para fazer a separação das fotos foi criado o algoritmo apresentado no Anexo II. O problema desse método é que preciso estipular um limiar para o valor de certeza do sistema retorna na classificação de cada imagem, isso porque que pretendemos pré-classificar novas imagens e elas podem conter exemplos de imagem que não pertence a nenhuma das classes do sistema. Então o foco desse experimento é achar um valor para esse limiar de forma empírica. Lembrando que o resultado dado pelo sistema não é unânime, ele apenas será uma ferramenta de auxílio para a classificação manual.

Para este experimento, foram apresentados 5 blocos de 100 fotos, onde as imagens poderia ser das três classes alvos deste trabalho ou nenhuma das três. Classificamos essas imagens que não pertence a nenhuma das classes como “outros”. Para cada bloco foi calculado o valor de limiar, para que assim, poderiamos usar a média destes valores como um valor próximo o ideal.

Para avaliar o desempenho, foi escolhido um limiar de 0.8 para a primeira execução do algoritmo. Depois da interação, foi feito um levantamento de quantas fotos foram classificadas corretamente, é analisado o quão distante do valor do limiar está do valor perfeito, a partir disso o valor do limiar é recalculado usando a equação a seguir:

Qc = Quantidade de imagens da classe “outros” nas classes reais do sistema
Qo = Quantidade de imagens de classe real do sistema classificada como “outros”.
Limiar[i] = Limiar atual

Limiar[i+1] = Limiar da próxima interação

Limiar[i+1] = Limiar[i] * (1 + (Qc - Qo) / Total de imagens).

Para este experimento, foram executadas algumas intenções até o valor do limiar se manter praticamente constante. Os resultados de cada interação de um dos experimentos está presente na tabela 4.4.

Interação	Limiar[i]	Qc	Qo	Limiar[i+1]
1	0.8	0	5	0.76
2	0.76	0	3	0.73
3	0.73	1	1	0.73
4	x	x	x	x

Tabela 4.4 - Valor do limiar

Podemos observar que depois do resultado da terceira iteração o valor do limiar não muda, dessa forma podemos dizer que para aquelas 100 imagens o valor ideal do limiar é de 0.73. Lembrando esse valor provavelmente irá mudar de base de dados para base de dados. Porém, esta é uma forma de aproximar o valor do limiar ideal para uma base de dados.

Como o objetivo do algoritmo apresentado no Anexo II é de apenas realizar uma pré-classificação, o fator erro é aceitável, pois todas as imagens ainda irão passar por uma classificação por um especialista.

Depois de realizar o mesmo experimento com os 5 blocos de 100 imagens diferentes, o valor do limiar final se manteve no intervalo entre 0.7 e 0.75.

4.5 Experimento 4 - Expansão da base de dados

Com os resultados do experimento 2, podemos observar que o sistema obteve melhores resultados com o crescimento da base de dados. Logo, este experimento vai avaliar qual o impacto no resultados com outra expansão da base de dados.

O maior problema de conseguir uma base de dados grande para esse trabalho é a classificação das imagens, pois ela foi feita manualmente. Porém com uma pré-classificação proposta pelo experimento 3, podemos levar em conta que o tempo de classificação manual seria reduzido.

Este experimento irá usar a pré-classificação sugerida pelo experimento 3 para a expansão da base de dados de 700 por classe para 1.500 por classe. Desta forma, podemos comparar os resultados obtidos por este experimento com o do experimento 2 e concluirmos se o sistema melhorou ou não.

Para isso foram baixadas novas imagens do instagram usando o 4k Stogram e foi feito uma pré-classificação usando o limiar 0.75. Depois das imagens pré-classificadas, foi feita uma classificação manual até que a base de dados atingisse o 1.400 imagens por classe, que no caso ficaram 980 para treinamento e validação e 420 para testes.

Depois disso foi executado o algoritmo de classificação (anexo I) e os resultados estão a seguir:

	Praia	Casa Noturna	Academia
Precisão	1	0.95	0.96
Recall	0.97	0.97	0.96
F1	0.98	0.96	0.96

Tabela 4.5 - Resultado do experimento 2

Nesse caso temos que manteve bons resultados para todas as classes para um teste de 420 imagens, dessa forma podemos afirmar que poder de abstração do sistema já possui um nível elevado.

4.6 Experimento 5 - Aumento na quantidade de épocas

Devido a quantidade de imagens no treinamento feito no experimento 4, podemos imaginar que a rede não teve a quantidade de épocas o suficiente para o melhor treinamento, dessa forma o experimento 5 aumentou a quantidade épocas

de 500 para 2.000 para servir de comparação se para esse caso a quantidade de épocas influência nos resultados.

Os resultados foram:

	Praia	Casa Noturna	Academia
Precisão	1	0.94	0.96
Recall	0.97	0.97	0.96
F1	0.98	0.95	0.96

Tabela 4.6 - Resultado do experimento 2

Podemos observar que os resultados praticamente se mantiveram o mesmo, logo o sistema consegue aprender de forma rápida e não é necessário um tempo extra no seu treinamento para o problema enfrentado neste trabalho.

4.7 Experimento 6 - Contagem de classes

Este experimento foi criado para a demonstração de um possível uso da solução desenvolvida pelo projeto. Neste caso, a demonstração do algoritmo demonstrado no Anexo III, que recebe um conjunto de imagens sem classe e retorna uma contagem de quantas imagens foram classificadas em cada classe. Para classificar como outros, foi usado o valor de limiar do experimento 3 (0.75) e o modelo gerado pelo experimento 5. Um exemplo de saída do algoritmo:

- praia : 21
- casa noturna : 12
- academia : 4
- other : 83

Para esse usuário analisado, podemos dizer que ele tem uma forte tendência em ir para praia, porém não muita para a academia. Podemos dizer que cerca de 10% das fotos recentes deste usuário são em casas noturnas. Essas informações são valiosas para os SR.

Desta forma os SR poderia trabalhar, tanto no algoritmo desse experimento quanto no seu uso. Como o algoritmo tem uma pasta alvo e sua resposta pode ser redirecionada para um arquivo de texto, os seus resultados já poderiam ser usados como uma classificação de ambiente por outros sistemas.

4.8 Disponibilidade de conteúdo

Como resultado técnico disponível dos resultados deste trabalho, tanto os três anexos quanto os modelos (rede treinada) resultante dos experimentos, estão disponíveis para download através do link: https://github.com/rlfscin/environment-classification

5 CONCLUSÃO E TRABALHO FUTUROS

Podemos observar que a solução dada para o problema de classificação apresentada neste trabalho, obteve uma taxa de acerto bem considerável. Devido à escolha do tema para obtenção das imagens ser “classes específicas do instagram”, a preparação dos dados se tornou bastante árdua. Porém, com o sistema já treinado, esse processo se torna muito mais fácil, como relatado no experimento 5. Este projeto confirmou que redes ConvNet têm resultados impressionantes com pouco esforço.

Apesar de as soluções deste trabalho não seja um algoritmo acoplável, as técnicas usadas aqui podem ser facilmente adaptadas a outros sistemas e o algoritmo em Anexo III pode ser usado como exemplo para tal. Também podemos levar em conta que ao final, o modelo (rede treinada) estará disponível e pode ser usada para classificações futuras por redes *Tensorflow*.

Em um caso específico, quando a imagem poderia ser classificada em duas classes, como por exemplo quando se tem uma pessoa malhando na praia. Nesse caso o sistema retorna as duas classes com taxas parecidas, mas baixas; logo, o algoritmo classificou como “outros”. Dessa forma, é preciso fazer um levantamento de como outros projetos de AM tratam esse problema para que as técnicas sejam aplicadas a este projeto. Outro ponto observado no decorrer dos experimentos, foi que o sistema tinha a capacidade de identificar objetos que pertenciam a algum ambiente, como esperado, sem nenhum tratamento especial para isso.

Um dos aprimoramentos deste trabalho seria desenvolver um algoritmo mais eficiente de melhor acoplamento para o uso dos SR. Esse novo sistema poderia receber uma chave de acesso à API do Instagram e conseguir as imagens para classificação direta no site.

Outro ponto seria a expansão das quantidades de classes para conseguir fazer uma melhor identificação das fotos que foram classificadas como “outras”. Essa melhoria seria mais fácil de fazer, porém, levaria um tempo, pois, para expandir o sistema, é preciso construir a base de dados inicial do sistema com a

ajuda de um especialista antes de poder usar o algoritmo mostrado no Anexo II para uma filtragem das imagens.

De modo geral, a escolha do DL para solucionar esse problema foi para mostrar como um método de AM, que está em evidência, se comportaria com a dificuldade, além de simplificar boa parte do trabalho do processo de processamento de imagem. Acredito que com a expansão da base de dados para várias classes, a rede DL irá manter seus bons resultados, diferentemente dos outros métodos de AM.

6 REFERÊNCIAS

Amo, S. (2014). Técnicas de Mineração de Dados. Universidade Federal de Uberlândia

Bouvrie, J. “Hierarchical Learning: Theory with Applications in Speech and Vision,” Ph.D. thesis, MIT, 2009

Braga, L. P. V. (2005). Introdução a Mineração de Dados. 2o Edição, Rio de Janeiro, E.paper

Deng L, 2014. A Tutorial Survey of Architectures, Algorithms, and Applications for Deep Learning. Microsoft Research ,Redmond, WA - USA

Felix, J. H. S (2007). Sistema de Visão Computacional para detecção e qualificação de ênfase pulmonar. Universidade Federal do Ceará

González, G., de la Rosa, J.L., and Montaner, M. (2007). Embedding Emotional Context in Recommender Systems. In The 20th International Florida Artificial Intelligence Research Society Conference-FLAIRS, Key West, Florida.

Han, J. Han, M. K. (2001). Data Mining: Concepts and Techniques. San Francisco, CA: Morgan Kaufmann.

Haykin, S. (2008) “Neural Networks and Learning Machines” 3rd Edition. Prentice Hall.

Hubel, D. and Wiesel, T. (1968). Receptive fields and functional architecture of monkey striate cortex. Journal of Physiology (London), 195, 215–243.

Joseph, B. Pine II (1993). Mass Customization. Harvard Business School Press. Boston, Massachusetts.

Kuncheva, L. K. (2004), “Combining Pattern Classifiers – Methods and Algorithms”.Wiley-Interscience.

Lin, T. Y., N. C. (1997). Rough Sets and Data Mining. Norwell, Massachusetts: Kluwer Academic Publishers.

Pereira, C. K., Campos, F., Ströele, V., David, J. M. N., Braga, R. (2014). Extração de Características de Perfil e de Contexto em Redes Sociais para Recomendação de Recursos Educacionais. Universidade Federal de Juiz de Fora.

Ren, X. Ramanan, D. Histograms of sparse codes for object detection. In CVPR, 2013.

Serre, T., Wolf, L., Bileschi, S., and Riesenhuber, M. (2007). Robust object recognition with cortex-like mechanisms. IEEE Trans. Pattern Anal. Mach. Intell., 29(3), 411–426. Member-Poggio, Tomaso.

Weiss, S. M. & Kulikowski, C. A. (1991). Computer systems that learn. San Mateo, CA: Morgan Kaufmann.

7 ANEXOS

7.1 Anexo I - Classificação

```
'''
Created by Rubens Lopes
This script is used to evaluate the system with the test files.
'''
import tensorflow as tf, sys, os
import numpy as np

test_path = 'teste Ex4'
label_path = 'test/retrained_labels.txt'
model_path = 'test/retrained_graphEx4.pb'

label_lines = [line.rstrip() for line
 in tf.gfile.GFile(label_path)]

with tf.gfile.FastGFile(model_path, 'rb') as f:
    graph_def = tf.GraphDef()
    graph_def.ParseFromString(f.read())
    _ = tf.import_graph_def(graph_def, name='')

config = tf.ConfigProto(allow_soft_placement=True)
with tf.Session(config=config) as sess:

     softmax_tensor =
sess.graph.get_tensor_by_name('final_result:0')
     dir_names = os.listdir(test_path)
     result = {}
     roc_data = {}
     num_exemples = 0

     for dir_class in dir_names:
          result[dir_class] = {}
          roc_data[dir_class] = []
          for key_class in dir_names:
               result[dir_class][key_class] = 0



     for dir_class in dir_names:

          class_name = dir_class
```

```
            data_loc = test_path + '/' + dir_class
            file_names = os.listdir(data_loc)
            test_size = len(file_names)
            num_exemples += len(file_names)

            for file_name in file_names:
                  image_path = data_loc + '/' + file_name

                  try:
                        image_data = tf.gfile.FastGFile(image_path,
'rb').read()
                        predictions = sess.run(softmax_tensor, \
                              {'DecodeJpeg/contents:0':
image_data})
                        top_k =
predictions[0].argsort()[-len(predictions[0]):][::-1]
                        prediction_class = label_lines[top_k[0]]
                        result[class_name][prediction_class] += 1
                        score = predictions[0][top_k[0]]

roc_data[class_name].append(np.array([top_k[0], score]))

                  # case the image is corrupted
                  except tf.errors.InvalidArgumentError:
                        os.remove(image_path)

      keys = []
      values = []
      for [key, value] in result.items():
            values.append(np.array([v for [k, v] in value.items()]))
            keys.append(key)
            print key + str(values[-1])

      values = np.array(values)
      print values

      for i in range(len(values)):
            print keys[i]
            precision = 1.0*values[i][i]/values[i].sum()
            recall = 1.0*values[i][i]/values[:,i].sum()
            f1 = 2*(precision*recall/(precision + recall))
            print 'Precisao: ' + str(precision)
            print 'Recall: ' + str(recall)
            print 'F1: ' + str(f1)
```

7.2 Anexo II - Separação de novos dados

```
'''
Created by Rubens Lopes
This script will separate a raw data into folders of predicted
classes
It will be used to filter raw data to help to label the data by
the specialist
'''
import tensorflow as tf, sys, os

threshold = 0.75

data_path = 'test/allData' #path of the raw data
label_path = 'test/retrained_labels.txt'
model_path = 'test/retrained_graphEx5.pb'

label_lines = [line.rstrip() for line
 in tf.gfile.GFile(label_path)]

with tf.gfile.FastGFile(model_path, 'rb') as f:
    graph_def = tf.GraphDef()
    graph_def.ParseFromString(f.read())
    _ = tf.import_graph_def(graph_def, name='')

config = tf.ConfigProto(allow_soft_placement=True)
with tf.Session(config=config) as sess:

     softmax_tensor =
sess.graph.get_tensor_by_name('final_result:0')

     file_names = os.listdir(data_path)
     for label in label_lines:
          os.mkdir(data_path + '/' + label)

     for file_name in file_names:
          image_path = data_path + '/' + file_name

          try:
               image_data = tf.gfile.FastGFile(image_path,
'rb').read()
               predictions = sess.run(softmax_tensor, \
                       {'DecodeJpeg/contents:0': image_data})
```

```
                top_k =
predictions[0].argsort()[-len(predictions[0]):][::-1]
                prediction_class = label_lines[top_k[0]]
                if predictions[0][top_k[0]] > threshold:
                     os.rename(image_path, data_path + '/' +
prediction_class + '/' + file_name)
                     print file_name + ' was predicted as ' +
prediction_class
                else:
                     print file_name + " wasn't prediceted as
anyclass"

           # case the image is corrupted
           except tf.errors.InvalidArgumentError:
                os.remove(image_path)
```

7.3 Anexo III - Contagens de imagens por foto

```
'''
Created by Rubens Lopes
This script will count how many image of each class there are in
the 'data_path' folder
'''
import tensorflow as tf, sys, os

data_path = 'test/counterData'
label_path = 'test/retrained_labels.txt'
model_path = 'test/retrained_graphEx5.pb'

label_lines = [line.rstrip() for line
 in tf.gfile.GFile(label_path)]

with tf.gfile.FastGFile(model_path, 'rb') as f:
    graph_def = tf.GraphDef()
    graph_def.ParseFromString(f.read())
    _ = tf.import_graph_def(graph_def, name='')

config = tf.ConfigProto(allow_soft_placement=True)
with tf.Session(config=config) as sess:

     softmax_tensor =
sess.graph.get_tensor_by_name('final_result:0')

     file_names = os.listdir(data_path)
```

```
	result = {}
	for label in label_lines:
		result[label] = 0
	result['other'] = 0

	for file_name in file_names:
		image_path = data_path + '/' + file_name
		try:
			image_data = tf.gfile.FastGFile(image_path,
'rb').read()
			predictions = sess.run(softmax_tensor, \
				{'DecodeJpeg/contents:0': image_data})
			top_k =
predictions[0].argsort()[-len(predictions[0]):][::-1]
			prediction_class = label_lines[top_k[0]]
			if predictions[0][top_k[0]] > 0.75:
				result[prediction_class] += 1
			else:
				result['other'] += 1

		# case the image is corrupted
		except tf.errors.InvalidArgumentError:
			os.remove(image_path)

	print 'count of each class'
	for [key, valeu] in result.items():
		print key + ' : ' + str(valeu)
```