UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Thiago Diniz da Silveira

RECONHECIMENTO DE MOVIMENTAÇÃO CORPORAL UTILIZANDO REDES NEURAIS

Florianópolis

2018

Thiago Diniz da Silveira

RECONHECIMENTO DE MOVIMENTAÇÃO CORPORAL UTILIZANDO REDES NEURAIS

Trabalho de conclusão de curso submetida ao Programa de graduação da Universidade Federal de Santa Catarina para a obtenção do Grau de Bacharel em Sistemas de Informação.
Orientador: Prof. Elder Rizzon dos Santos, Dr.

Florianópolis

2018

Catalogação na fonte elaborada pela biblioteca da
Universidade Federal de Santa Catarina

A ficha catalográfica é confeccionada pela Biblioteca Central.

Tamanho: 7cm x 12 cm

Fonte: Times New Roman 9,5

Maiores informações em:
http://www.bu.ufsc.br/design/Catalogacao.html

Thiago Diniz da Silveira

RECONHECIMENTO DE MOVIMENTAÇÃO CORPORAL UTILIZANDO REDES NEURAIS

Esta Trabalho de conclusão de curso foi julgada aprovada para a obtenção do Título de "Bacharel em Sistemas de Informação", e aprovada em sua forma final pelo Programa de graduação da Universidade Federal de Santa Catarina.

Florianópolis, 01 de julho 2018.

Banca Examinadora:

Prof. Renato Cislaghi, Dr.
Presidente

Prof. Elder Rizzon dos Santos, Dr.
Orientador

Prof. Jerusa Marchi, Dr(a).

Prof. Thiago Angelo Gelaim, Msc

Dedico esse trabalho a minha família que sempre me apoiou e quis muito que eu terminasse a faculdade. Também a minha noiva que sempre me deu suporte em momentos difíceis ao longo desse trabalho.

AGRADECIMENTOS

Agradeço ao meu pai e minha mãe por todas as condições que me deram para estudar e conseguir terminar a faculdade. Também gostaria de agradecer as minhas irmãs que sempre me apoiaram e me ajudaram na parte de revisar esse trabalho. Gostaria de agradecer a minha noiva por ter feito eu ter vontade de novo de me formar e obter uma graduação. Agradeço aos meus professores por terem sido tão complacentes e ter paciência comigo.

A persistência é o menor caminho do êxito.

Charles Chaplin

RESUMO

Redes neurais artificiais são modelos computacionais que imitam o sistema nervoso central de um humano. Hoje muitas empresas estão utilizando inteligência artificial em áreas distintas como: carros autônomos, segurança, jogos e em até ferramentas que auxiliam pessoas com deficiências motoras. Este trabalho visa apresentar um modelo de reconhecimento de movimentação corporal utilizando redes neurais, fazer análises e teste do mesmo. Foram estudados quatro modelos: *Probabilistic High-Dimensional Regression, Hyperface, Deepgaze e Vanilla CNN*. Os dois primeiros foram implementados. *Probabilistic High-Dimensional Regression* foi implementado em Matlab e o *Hyperface* em Python. Por fim foi escolhido o modelo *Hyperface* para serem feitos os testes e as análises. O resultado obtido foi satisfatório de forma geral, porém foi percebido que muito da análise tem interferência pelo dataset e o treinamento utilizado para o aprendizado do modelo.

Palavras-chave: Redes Neurais, Inteligência Artificial, Redes Neurais Convolucionais, *Hyperface*.

ABSTRACT

Artificial neural networks are computational models that imitates the central nervous system of a human. Today many companies are using artificial intelligence in areas such as: autonomous cars, security, games and even tools that help people with motor disabilities. This work aims to present a model of recognition of body movement using neural networks, make analyzes and test it. Four models were studied: Probabilistic High-Dimensional Regression, Hyperface, Deepgaze and Vanilla CNN. The first two were implemented. Probabilistic High-Dimensional Regression was implemented in Matlab and Hyperface in Python. Finally, the Hyperface model was chosen for testing and analysis. The result was satisfactory in general, but it was noticed that much of the analysis has interference by the dataset and the training used to learn the model.

Keywords: *Neural Networks*, *Artificial Intelligence*, *Neural Networks Convolutionals*, *Hyperface*.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SUMÁRIO

1 INTRODUÇÃO

A Inteligência Artificial (IA) começou a ser estudada após a segunda guerra mundial e segundo Russel e Norvig (2004, p. 2) é citada regularmente como "o campo em que eu mais gostaria de estar" por cientistas de outras áreas. Surgiram vários trabalhos que poderiam ser considerados como IA entre a década de 40 e 50, porém apenas no artigo *"Computing Machinery and Intelligency"* feito por Alan Turing em 1950, é que foi articulada uma visão completa da IA. Nesse artigo Turing (1950) apresentou o teste de Turing, aprendizagem de máquina, algoritmo genéticos e aprendizagem por reforço.

Entretanto foi no verão de 1956 em um seminário realizado em Darthmouth que essa área de estudo se tornou conhecida como Inteligência Artificial, nome este sugerido pelo pesquisador John McCarthy.

Dentro da inteligência artificial existem dois paradigmas mais representativos que são o simbólico, que segundo Russel e Norvig (2004, p. 20) foi apresentado por Newell e Simon em 1976. Esta representação utiliza os símbolos e outros conhecimentos explícitos, eles afirmam que todo sistema que tenha inteligência deve operar com uma estrutura de dados composta por símbolos - e conexionista - que representam o raciocínio com conexões como as redes neurais, que apresentam os neurônios e suas conexões como forma de racionalizar. Apesar das suas diferentes abordagens Russel e Norvig afirmam que elas devem ser vistas como complementares e não como concorrentes (2004, pg. 26).

O primeiro trabalho que hoje é reconhecido como IA Conexionista, conforme Russel e Norvig (2004, p. 18), foi realizado por Warren McCulloch e Walter Pitts em 1943, os dois pesquisadores propuseram um modelo de neurônios artificiais e mostraram que qualquer função computável podia ser calculada por uma rede de neurônios conectados, além disso sugeriram também que a rede seria capaz de aprender. Esse modelo de neurônios artificiais hoje em dia é conhecido como redes neurais artificiais (RNAs) e como o conhecimento é representado por conexões então ela é da abordagem conexionista e é sobre isso que iremos focar nosso estudo.

Haykin (2001, p. 1) apresenta Redes Neurais como uma tecnologia que tem raízes em muitas disciplinas: neurociência, matemática, estatística, física, ciência da computação e engenharia. Possui uma propriedade muito importante que é a de aprender. Uma rede neuronal ou neural tem o intuito de representar como o cérebro humano realiza uma tarefa particular ou função de interesse.

Redes Neurais Artificiais são desenvolvidas com componentes eletrônicos ou então simuladas com programação em um computador. Uma RNA é composta de neurônios que são interligados entre eles, as conexões entre esses neurônios são chamadas de sinapses.

De acordo com Coppin (2013, p. 233) aprendizagem é um segmento extremamente importante da IA. Coppin (2013, p. 248) afirma que as abordagens mais relevantes de aprendizagem são por reforço, supervisionada e a não supervisionada.

Quando um sistema é submetido a aprendizagem por reforço, ao executar uma ação com sucesso, ele recebe um reforço positivo, caso contrário irá receber um reforço negativo, por exemplo um agente jogador de damas, ao mover uma peça, se o agente capturar uma peça adversária este recebe um reforço positivo senão um negativo. A aprendizagem supervisionada é onde as redes aprendem ao serem apresentadas aos dados pré-classificados, uma rede neural com esse tipo de aprendizagem pode ajustar os pesos das conexões dos neurônios para obter melhor precisão da informação com intervenção humana. Por último temos a aprendizagem não supervisionada, a qual não há intervenção humana, um exemplo de rede neural que utiliza esse tipo de aprendizagem é o Mapa de Kohonen que é capaz de aprender a classificar um conjunto de dados sem saber quais são as classificações ou sem receber quaisquer dados de treinamento.

Neste projeto usaremos os dados obtidos através das gravações realizadas pela Universidade de Salford[1] para então fazermos o reconhecimento de movimentação corporal da pessoa que aparece no vídeo.

Os vídeos foram gravados num ambiente de realidade virtual, onde uma pessoa manipula um semáforo utilizando um celular. O propósito dos vídeos é para um projeto que está analisando o fator de distração de uma pessoa ao manipular um smartphone e a sua relação aos acidentes de trânsitos. O problema foi reconhecido pela Organização Mundial da Saúde como um fator de risco no relatório mundial de prevenção de acidentes.

A finalidade deste trabalho de conclusão de curso é reconhecer a movimentação corporal de uma pessoa que apareça no vídeo, o reconhecimento da movimentação será focado na direção do rosto da pessoa. Faremos testes com uma rede neural convolucional ou também conhecida como convolutiva, esse tipo de rede neural utiliza a aprendizagem supervisionada. Esse modelo segundo Haykin (2001, p. 271) serve para reconhecer formas bidimensionais com um alto grau de invariância quanto formas de distorções e por isso é utilizado para reconhecimento

[1] Universidade de Salford - https://www.salford.ac.uk/

de imagens.

1.1 OBJETIVOS

Esta seção tratará dos objetivos gerais e específicos a serem alcançados por este Trabalho de Conclusão de Curso.

1.1.1 Gerais

Desenvolver um modelo computacional que permita reconhecer movimentações corporais em vídeo.

1.1.2 Específicos

1. Analisar os diferentes métodos computacionais de aprendizagem para reconhecimento de movimentação corporal;
2. Delimitar o escopo dos movimentos a serem reconhecidos;
3. Propor um modelo computacional para reconhecimento de movimentação corporal;
4. Treinar o modelo computacional escolhido;
5. Testar e validar o modelo computacional.

1.2 ORGANIZAÇÃO

1.2.1 Descrição

Este trabalho será desenvolvido em parceria com a Universidade de Salford e contará com dados de pesquisas obtidos em suas instalações para testar a Rede Neural e chegar aos resultados esperados.

1.2.2 Método de Pesquisa

A natureza da pesquisa utilizada será Exploratória, na qual será estudado qual tipo de Rede Neural a ser utilizado, depois procurare-

mos *frameworks opensource* adequado para a criação da Rede Neural e faremos um caso de uso para podermos realizar o treinamento da rede neural, os dados serão provenientes da Universidade de Salford.

1. Analisar os diferentes métodos computacionais de aprendizagem para reconhecimento de movimentação corporal

   Para conhecer os métodos computacionais de aprendizagem foi realizado a leitura dos livros: Inteligência Artificial de (COPPIN, 2013), Inteligência Artificial (RUSSEL; NORVIG, 2004) e Redes Neurais (HAYKIN, 2001). Além disso foi feito pesquisas na internet e leitura de TCC's antigos.

   Podemos encontrar os métodos estudados na seção 3.

2. Delimitar o escopo dos movimentos a serem reconhecidos

   Na seção 2.2 foi definida as posições mais relevantes de movimentos para o projeto.

3. Propor um modelo computacional para reconhecimento de movimentação corporal

   Serão estudados os modelos computacionais de aprendizagem, dentre os quais iremos escolher um modelo para reconhecimento de imagens. Os modelos pesquisados estão na seção 4.1.

4. Treinamento do modelo computacional escolhido

   Treinaremos nossa rede neural, de forma que tentaremos chegar ao resultado esperado. Como foi visto na seção 4.2.2.3.

5. Testar e validar o modelo computacional

   Os dados resultantes devem coincidir com os dados da Universidade de Salford. Resultados e Análises podem ser vistos na seção 5.

2 RECONHECIMENTO DE MOVIMENTAÇÃO CORPORAL

O cérebro humano é considerado um computador altamente complexo, não-linear e paralelo. Ele tem a capacidade de processar informações (por exemplo reconhecimento de padrões, percepção e controle motor) muito mais rápido que qualquer computador hoje em dia devido a utilização dos seus neurônios e sinapses. (HAYKIN, 2001, p. 27).

O neurônio, segundo Coppin (2013), é um elemento de processamento que é composto de uma soma, que é o corpo do neurônio, um axônio e vários dendritos, como podemos ver na Figura 1 .

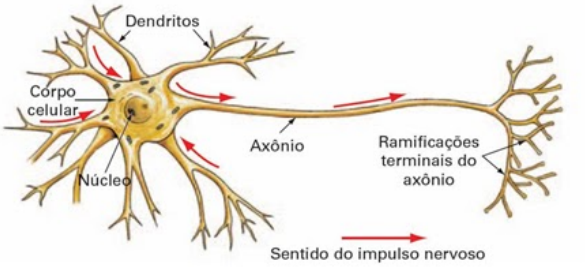


Figura 1 – Neurônio

A comunicação entre neurônios ocorre da seguinte maneira. Primeiro um neurônio recebe um sinal de entrada através dos dendritos vindo de outros neurônios, quando esse sinal excede um limiar, então esse neurônio é ativado e envia seu sinal para outro neurônio através das ramificações terminais do axônio. Na computação é a mesma coisa. (COPPIN, 2013, p. 254).

O neurônio artificial por sua vez recebe uma série de entradas, uma função conhecida como função de ativação é aplicada no valor resultante do somatório dessas entradas e que resulta em um nível de ativação do neurônio, que será o valor de saída desse neurônio.

A modelagem de uma rede neural geralmente é realizada em multi-camadas, isso quer dizer, que terá uma camada com vários neurônios, e cada neurônio irá fazer conexão com os neurônios da outra camada, e assim sucessivamente até a camada de saída. (COPPIN, 2013, p. 262).

Para reconhecer a movimentação corporal, que é um desafio complexo computacionalmente porém simples para um cérebro humano, primeiramente precisamos reconhecer um humano. Nguyen, Li e Ogunbona (2016) apud Wang (2017) citam que geralmente o processo de detecção de humano é através dos seguintes passos: extrair regiões de interesse de uma imagem, descrever as regiões com descritores e então classificá-las como humana ou não-humana.

2.1 APRENDIZADO SUPERVISIONADO

Coppin (2013) cita que uma propriedade importante do cérebro humano é a plasticidade, isso quer dizer que um neurônio é capaz de fazer conexões com outros neurônios e alterar a sua natureza conforme aos eventos que ocorrem, com isso o cérebro é capaz de aprender. Haykin (2001) afirma que a propriedade primordial para uma RNA é a mesma habilidade citada por Coppin. Em uma Rede Neural Artificial, o aprendizado ocorre pela alteração dos pesos entre as conexões dos neurônios.

Existem os aprendizados supervisionados e não supervisionados, contudo iremos nos aprofundar a qual utilizamos nesse trabalho que é a aprendizagem supervisionada.

Quando uma rede neural é configurada para ter um aprendizado supervisionado, Haykin (2001) explica que em virtude de um conhecimento prévio, o professor pode dar à rede neural o resultado esperado, ou seja, o treinamento supervisionado precisa de um conjunto de dados já previamente classificado.

Inicialmente a rede neural poderá retornar respostas completamente diferentes dos resultados esperados, se dermos como entrada um vetor para a rede neural, o professor saberá previamente o resultado desse vetor, e a rede neural será ajustada alterando o peso das conexões entre os neurônios conforme o erro que obtiver comparado ao resultado ótimo (do professor). Podemos ver um diagrama na Figura 2 demonstrando o comportamento de um treinamento.

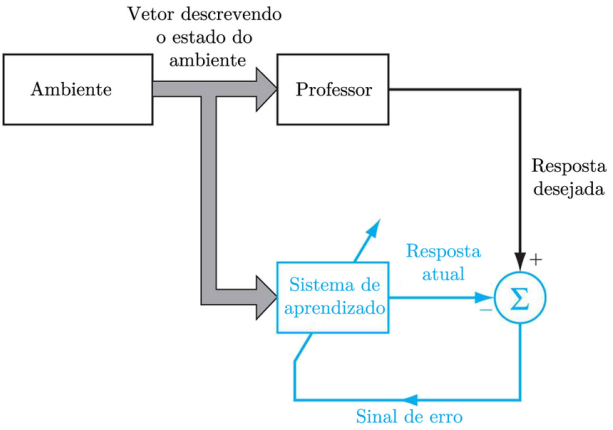


Figura 2 – Aprendizado Supervisionado

A diferença das respostas é conhecida como sinal de erro e é utilizada em conjunto com o vetor de entrada para corrigir o peso sináptico de cada neurônio de forma interativa.

O processo é repetido até que a RNA obtenha a resposta em comparação do professor com o erro médio quadrático (em inglês MSE, *Mean Squared Error*) inferior a um limite que foi previamente estabelecido.

Para calcular o MSE utilizamos a expressão $MSE = \frac{1}{N}\sum_{i=1}^{N}(e_i)^2$, onde, N é o número de saídas consideradas, (e_i) a diferença entre a i-ésima saída da rede e a sua correspondente função alvo, que é a função de ativação do neurônio.

2.1.1 Redes convolucionais

Redes neurais convolucionais (RNC) são um tipo específico de rede neural artificial, segundo Bengio, Goodfellow e Courville (2015) apud Wang (2017) no artigo *Deep Learning* em 2016, trata-se de usar convolução em pelo menos uma camada ao invés de operações com vetores e matrizes. Inicialmente a RNC foi criada para problemas de classificação de imagem.

Em 1998 foi criado um protótipo de RNC conhecida como LeNet, que é uma rede com 7 camadas, incluindo 3 convolucionais e uma camada totalmente conectada e foi utilizada para reconhecer escritas a mão livre segundo LeCun (1998). Porém em 1998 os recursos compu-

tacionais não eram tão avançados, então apenas em 2012 com a criação da AlexNet quando obteve um grande sucesso no concurso *ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) 2012*, os métodos para redes neurais convolucionais conseguiram ser usados por todos.

Alexnet seguiu principalmente a ideia da LeNet porém aumentou para 5 camadas convolucionais e 3 totalmente conectadas como podemos ver na Figura 3.

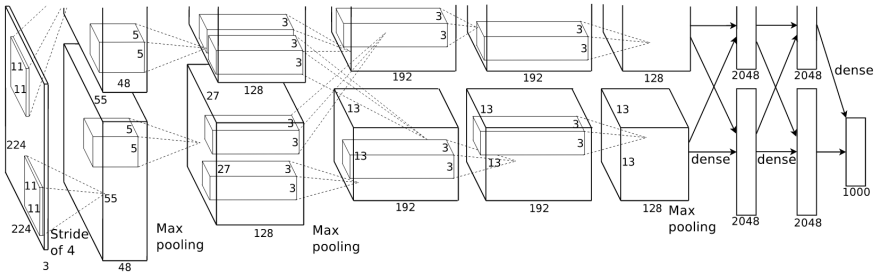


Figura 3 – Alexnet

Um processo típico de uma camada convolucional inclui convolução, função de ativação e uma fase de agrupamento. Um exemplo de uma camada convolucional é apresentado na Figura 4:

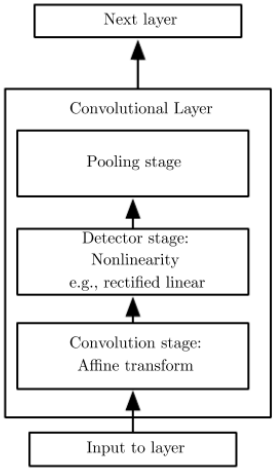


Figura 4 – Camada Convolucional

Em camadas convolucionais inferiores as funcionalidades aprendidas são simples, como linhas e gradientes, mas nas camadas superiores funcionalidades mais complexas são aprendidas, como textura e formas complexas.

Uma operação de convolução geralmente é seguida por uma função não linear de ativação que frequentemente é uma unidade linear retificadora (ULR) ou inglês (ReLU) nas RNCs. Nair e Hinton (2010) apud Wang (2017) afirmam que a ULR é dada pela função $f(x) = max(0, x)$ como estado de entrada desde que a ULR descarta negativas respostas de uma entrada. Isso reduz a taxa de ativação em 50%. *Pooling layers* também são importantes, são as camadas de agrupamento e é inserida entre as operações de convolução.

Geralmente a estrutura de uma RNC é seguida por uma camada totalmente conectada. Isso faz com que o mapeamento convolucional se transforme em camadas de rede neural normal.

Assim como as RNAs, as RNCs também são treinadas utilizando *back propagation* minimizando a função de perda com um otimizador como descendente de gradiente estocástico (DGE). Como a rede neural fica grande o processo de treinamento é bem complexo computacionalmente. Um exemplo é a AlexNet. O modelo tem 8 camadas capazes de aprender e foi treinada com 60 milhões de parâmetros e 650000 neurônios, levando alguns dias executando com uma paralelização em GPU para completar o treinamento.

2.2 POSIÇÃO DA CABEÇA

Como foi falado nos objetivos específicos, foi delimitado uma região para o reconhecimento da movimentação corporal. Neste trabalho foi escolhido a região da cabeça.

Foi utilizado a anotação *pitch, yaw, roll* comumente usada para descrever os três graus de liberdade da posição da cabeça. Conforme a figura 5 *Pitch* e *Yaw* indicam o deslocamento da posição da cabeça entre horizontal e vertical e *Roll* é a angulação.

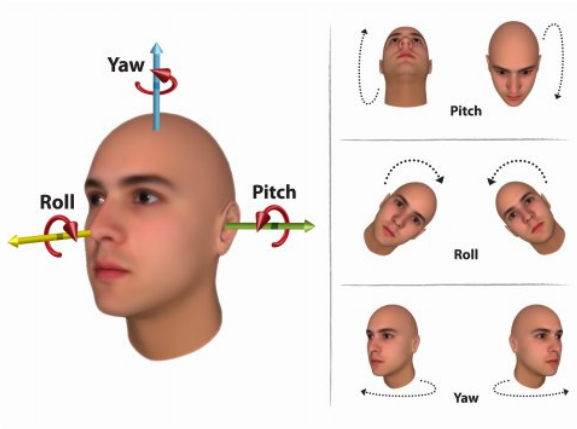


Figura 5 – Pitch, Yaw e Roll

3 TRABALHOS RELACIONADOS

Para pesquisar os modelos de rede neural que fazem reconhecimento de movimentação corporal foram feitas pesquisas no Google Scholar e no Research Gate por ordem de data, a intenção era buscar os artigos mais recentes que abordaram esse assunto. Foram encontrados artigos sobre quatro modelos, que foram: *Probabilistic High-Dimensional Regression* criado por Drouard et al. (2015), Hyperface formulado por Ranjan, Patel e Chellappa (2016), Deepgaze citado por Patacchiola M. (2017), *Vanilla CNN* estudado por Wu et al. (2016).

3.1 HYPERFACE

Hyperface segundo Ranjan, Patel e Chellappa (2016) é um algoritmo que usa rede neural convolucional para prover simultâneas funcionalidades como: detectar faces, localização de pontos relevantes da face, estimação da pose da face e reconhecimento de gênero em uma foto.

Os estudos feitos pelos idealizadores do *Hyperface* provam que um algoritmo com múltiplas funcionalidades tem uma performance melhor do que um algoritmo específico para cada uma das funcionalidades citadas.

O algoritmo é uma mistura dos trabalhos: *Face detection, pose estimation, and landmark localization in the wild. In IEEE Conference on Computer Vision and Pattern Recognition* criado por Zhu e Ramanan (2012) e da sua extensão *FaceDPL: Detection, pose estimation, and landmark localization in the wild.*.

São três módulos que compõem a *Hyperface*, o primeiro cria regiões independentes da foto analisada e reescala essa região para 227 x 277 pixels. O segundo módulo é uma RNC que utiliza essa foto reescalada e classifica como face ou não face. Caso a região da imagem seja classificada como face, então a rede adiciona as marcações faciais, estimação da pose da face e informações do gênero. O terceiro módulo é um passo de pós processamento que utiliza as informações geradas no módulo dois para dar uma nota e melhorar a performance das tarefas individuais.

3.1.1 Arquitetura

Inicialmente a *Hyperface* é composta por uma rede neural conhecida como AlexNet para classificação de imagem. Esta foi modificada para conter apenas 5 camadas convolucionais como podemos ver na Figura 6. A rede é inicializada com os pesos da rede RCCN_Face citada no artigo *Rich feature hierarchies for accurate object detection and semantic segmentation* (GIRSHICK et al., 2014) e treinada para a tarefa de detecção de faces.

Após o processamento é feita uma fusão das camadas da AlexNet para no final termos 5 camadas de saída representando: Detecção de face (face ou não face), Marcações faciais (pontos), Visibilidade (fator), Pose (*roll, pitch e yaw*) e Gênero (masculino ou feminino).

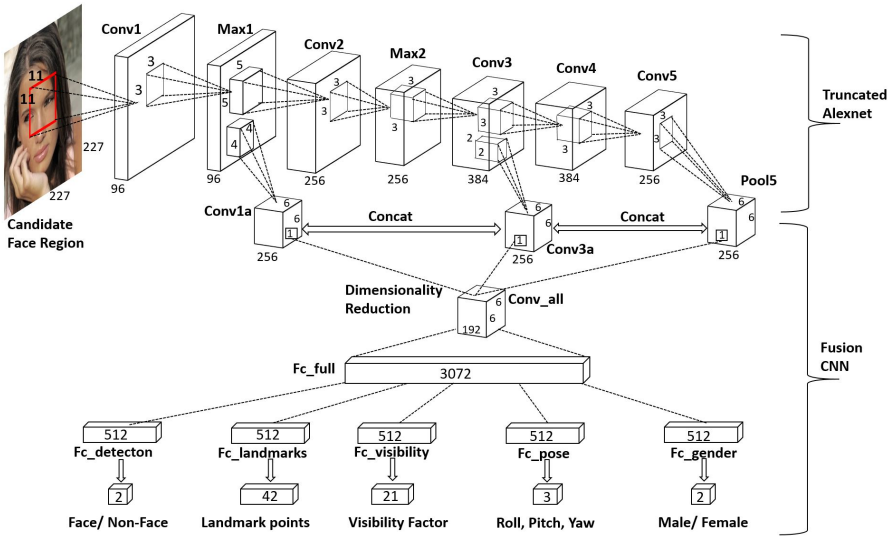


Figura 6 – Arquitetura Hyperface

3.1.2 Treinamento e AFLW Dataset

Para o treinamento da Hyperface (RANJAN; PATEL; CHELLAPPA, 2016) foi utilizado o *AFLW Dataset*, é um *dataset* que contém 25.993 imagens com marcações faciais, onde cada imagem tem 21 marcações. As imagens possuem variações de pose, expressão, etnia, idade e gê-

nero. Foram utilizadas 1000 imagens para teste, e as outras para o treinamento da rede.

Para a estimativa da posição da pose da face foi utilizada a perda Euclidiana para estimar a roll (p1), pitch (p2) e yaw (p3), foi computado a perda por cada região da imagem obtendo uma sobreposição maior que 0.5.

$lossp = \frac{(\hat{p}1-p1)^2+(\hat{p}2-p2)^2+(\hat{p}3-p3)^3}{3}$

Como podemos ver na função descrita, o valor resultante de *loss* é dado pela média dos ângulos que são dados pela diferença elevado ao quadrado entre o valor de $\hat{p}$ estimado e do valor de p real.

3.1.3 Teste

Para cada imagem é extraída uma região, e apenas essas subimagens geradas que possuem uma pontuação maior que um limite pré definido são classificadas como tendo face e serão processadas.

3.1.4 Resultado

A *Hyperface* foi comparada com a *RCCN_ Pose* e *Multitask_ Face*. A conclusão que Ranjan, Patel e Chellappa (2016) chegaram é que quando tem uma rede neural com múltiplas funções que tem conexão, elas têm uma melhor taxa de acerto do que uma rede neural com apenas uma única função.

3.2 MODELO PROBABILISTIC HIGH-DIMENSIONAL REGRESSION

Estimar a posição da cabeça é importante para diversos cenários como anáÍise de eventos sociais, interação entre robô e humano e também como sistema de assistente de motorista. A posição geralmente é expressa em três ângulos (*pitch, yaw, roll*) e este trabalho propõe uma solução utilizando o modelo de Mistura Gaussiana de mapeamento de localização linear (GLLiM).

O trabalho desenvolvido por Drouard et al. (2015) tem que resolver um problema de mapeamento de *high-dimensional* para *low-dimensional*, o *high-dimensional* é bem desafiador pois há muitos parâmetros que devem ser estimados, então foi proposta a GLLiM que faz uma regressão *high-to-low* que aprende uma regressão *low-to-high*

e então é inferida uma predição *high-to-low* baseado na inversão de Bayes.

A vantagem desse modelo é que diminui a quantidade de parâmetros que devem ser estimados e que estão associados com o aprendizado *high-to-low*.

O funcionamento do modelo é, dada uma região com o rosto, um mapeamento é construído entre a as posições da cabeça e a região dos descritores baseado em *Histogram of Oriented Gradients* (HOG) que é uma funcionalidade utilizada em processamento de imagem para detecção de objetos.

3.2.1 Predição iterativa com GGLiM

O modelo assume que uma caixa delimitadora é centralizada perfeitamente sobre a face. Na prática os detectores de rostos produzem erros, e para compensá-los nós consideramos uma variável x onde x é um vetor de poses.

Foi treinado o mapeamento do descritor baseado em HOG para o aumento do vetor x. Permitindo que o modelo preveja tanto a posição da face como uma imagem de compensação.

Se essa predição for aplicada várias vezes, o posicionamento da caixa delimitadora é refinada e eventualmente convergindo para um alinhamento perfeito entre a previsão da caixa delimitadora e a verdadeira caixa delimitadora.

3.2.2 Datasets

Para fazer os testes foram utilizados dois datasets: *Prima head pose image dataset* e *Biwi Kinect head pose dataset*. O primeiro contém imagens e o segundo vídeos e os dois datasets são anotados com os ângulos em graus da pose e posição do rosto para cada imagem e frame. A pose x é expressa em dois ângulos no Prima (yaw e pitch) e em três no Biwi (yaw, pitch e roll).

3.2.3 Resultados

Segundo Drouard et al. (2015) o modelo proposto por eles, teve um desempenho melhor que os modelos Seemann, Nickel e Stiefelhagen (2004), Gourier, Hall e Crowley (2004) e Ricci e Odobez (2009), obtendo

menos erros, tanto com os rostos à mostra quanto escondidos.

4 DETECÇÃO DE DIRECIONAMENTO EM VÍDEO

Esta seção explica os objetivos específicos de como escolhemos um modelo de rede neural, o treinamento, o teste e a validação da mesma. Conforme a Figura 7 podemos ver que dividimos o trabalho em duas raias principais, desenvolvimento e utilização. No desenvolvimento abordamos o objetivo de treinamento do modelo selecionado. Na raia de utilização foram realizados testes e validações com vídeos para analisar o resultado do treinamento.

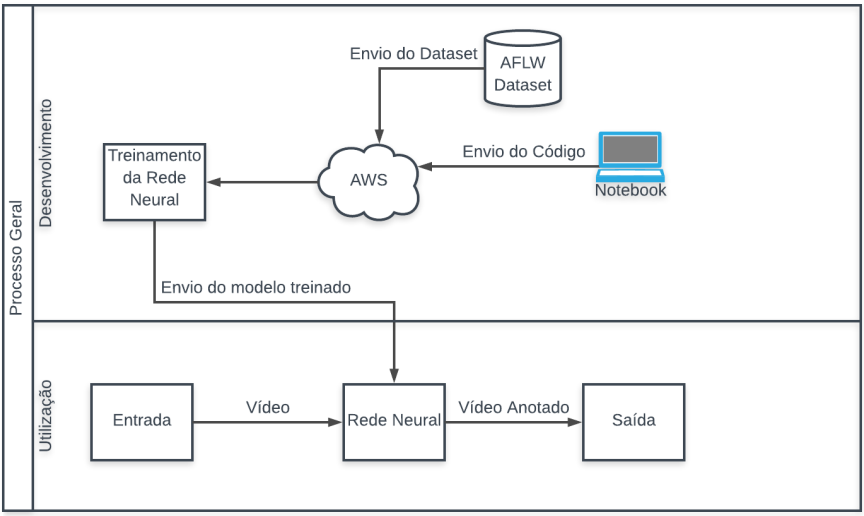


Figura 7 – Diagrama Geral

4.1 SELEÇÃO DOS MODELOS

Os modelos *Hyperface* e *Probabilistic High-Dimensional Regression* são mais específicos e alinhados ao objetivo geral do trabalho que é o estudo de reconhecimento de movimentação corporal. Face ao exposto foram implementados esses dois modelos.

4.2 ANÁLISE DOS RESULTADOS

A análise do resultado de um modelo é feita para cada frame de um vídeo. Foi utilizado a anotação *pitch, yaw, roll* explicada na seção 2.2.

Na figura 8 essa triangulação entre *roll, yaw e pitch* está localizada no canto inferior esquerdo, a qual utilizaremos para saber se a rede neural acertou ou errou a análise.

Além disso como utilizamos a rede neural *Hyperface*, temos outros dados que vamos aproveitar para analisar. No canto superior direito há uma bolinha que quando está azul quer indicar que o sexo da pessoa é masculino ou vermelha indicando que é feminino. Também temos os pontos faciais, que quando estão em verde indicam que foram encontradas todas as anotações faciais.

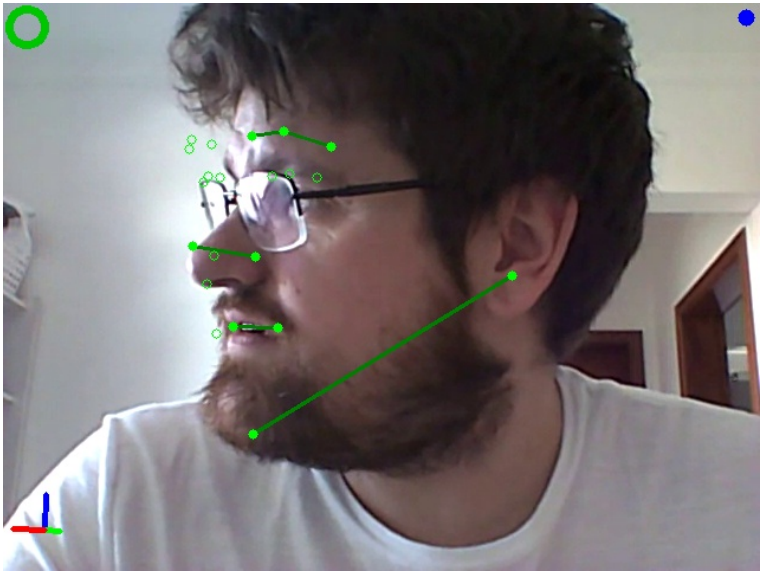


Figura 8 – Pitch, Raw e Roll na implementação

4.2.1 Datasets

A análise de um vídeo é feita a cada frame como mencionamos anteriormente, então o treinamento da rede neural é feito por um data-

set composto por imagens. Contudo, o objetivo é fazer reconhecimento de movimentação corporal em vídeos, à vista disso os datasets utilizados para fazer as análises foram obtidos através de vídeos gravados pela webcam, vídeos obtidos do YouTube e também vídeos gravados na Universidade de Salford.

4.3 MODELOS DE RECONHECIMENTO DE MOVIMENTAÇÃO CORPORAL

Nesta seção será explicado os modelos *Probabilistic High-Dimensional Regression* (DROUARD et al., 2015) e *Hyperface* (RANJAN; PATEL; CHELLAPPA, 2016).

4.3.1 Modelo Probabilistic High-Dimensional Regression

Inicialmente foi analisado o modelo *Probabilistic High-Dimensional Regression* detalhado na seção 3.2, que não utiliza redes neurais, porém não foi aprovado em virtude de diversos problemas de performance apresentados ao rodar o modelo no software Matlab. Para a implementação foi utilizado o *Prima Dataset* criado por Gourier D. Hall (2004) que contém 2790 imagens de 15 pessoas com ângulos diferentes da posição da cabeça. Como mencionamos em 4.1.2.1 os datasets de treinamento são compostos por imagens, pois a análise de um vídeo é feita através de cada frame.

Podemos ver uma parte do código fonte referente a parte de processamento do dataset. O código é auto explicativo, pegamos as imagens do dataset e suas marcações e retornamos o valor em um array.

```
1  function data = processPrima(data_path)
2   sample_index = 1;
3    for f = 1:15 % loop over all the person in the database
4     fprintf('ID %02d\n',f);
5     images = dir(['dados/Person' sprintf('%02d',f) '/*.jpg']);
6     reverseStr = [];
7     for i = 1:length(images)% loop over all images of a person
8      % load images
9      frame = imread(['dados/Person' sprintf('%02d',f) '/' images(i).name]);
10     id = fopen(['dados/Person' sprintf('%02d',f) '/' images(i).name(1:end-3) 'txt'
11     st=fread(id,'*char');
12     fclose(id);
```

```
13      % extract booding box related information
14      tmp = str2num(st(28:end-1));
15      bbox = [tmp(1)-tmp(3)/2;tmp(2)-tmp(4)/1.5;tmp(1)+tmp(3)/2;tmp(2)+tmp(4)/1.5];
16      bbox(1) = max(1,bbox(1));
17      bbox(2) = max(1,bbox(2));
18      bbox(3) = min(size(frame,2),bbox(3));
19      bbox(4) = min(size(frame,1),bbox(4));
20      face = frame(round(bbox(2)):round(bbox(4)),round(bbox(1)):round(bbox(3)),:);
21      face = histeq(rgb2gray(imresize(face,[64 64])));% resize and convert to grayscale
22      hog = [HOG(face,8,[32 32],[2 2]);...
23          HOG(face,8,[16 16],[2 2]);...
24          HOG(face,8,[8 8],[2 2])];% compute feature vector
25      data.features(:,sample_index)=[sample_index;hog];
26      p=images(i).name(12:end-4);
27      a=find(p=='+' | p=='-');
28      data.pose(:,sample_index) = [
29          sample_index;str2double(p(a(1):a(2)-1));
30          str2double(p(a(2):end))
31      ];
32      % save information
33      data.box(:,sample_index) = [sample_index;bbox];
34      data.index(:,sample_index) = [sample_index;f];
35      data.image_path(:,sample_index) = {sample_index,[
36          'dados/Person' sprintf('%02d',f) '/' images(i).name
37      ]};
38      sample_index = sample_index +1;
39      % display
40      msg = sprintf('Processed %d/%d', i, length(images));
41      fprintf([reverseStr, msg]);
42      reverseStr = repmat(sprintf('\b'), 1, length(msg));
43      end
44     fprintf('\n');
45   end
46  end
```

Face ao exposto, outras alternativas foram procuradas, resultando no modelo *Hyperface* apresentado na seção 4.3.2.

4.3.2 Modelo Hyperface

A implementação desse algoritmo foi realizada em Python que é uma linguagem de programação de alto nível sendo muito utilizada

para processamento de imagens e inteligência artificial.

4.3.2.1 Código fonte utilizado como base

Utilizamos como base para o desenvolvimento um código fonte disponibilizado no GitHub por TAIKU (2017), que implementa o *Hyperface*, porém sem aceitar entradas em formato de vídeo. O trabalho consistiu em alterar o código para utilizar bibliotecas mais novas, pois o servidor que criamos na Amazon para fazer o treinamento que será explicado na seção 4.3.2.3 já estava configurado com a versão do Chainer 3. Além da atualização das bibliotecas adicionamos a opção de processamento em vídeo.

4.3.2.2 Dados utilizados para treinamento

Os dados utilizados para treinamento da RNA foram aqueles proveniente do AFLW *Dataset* (KOESTINGER PAUL WOHLHART; BISCHOF, 2011). Este *dataset* tem um total de 25 mil faces com variações de etnia, poses, expressões, idade e gênero. Todas as imagens tem anotações com 21 pontos de marcações faciais. O ideal seria utilizar um *dataset* de treinamento criado por nós, porém não seria possível terminá-lo em tempo, pois para um vídeo deveríamos pegar cada *frame* do mesmo e fazer as anotações faciais, de gênero e a posição da cabeça anotando o *pitch*, a *raw* e o *roll*.

4.3.2.3 Treino

Devido ao fato do computador que utilizamos localmente não ter memória suficiente, ao executar o algoritmo de treinamento capturamos muitas exceções de falta memória. Em consequência optamos por treinar a RNA em ambiente externo. O treinamento da rede neural foi executado em um servidor na Amazon c3.x2large que estava previamente configurado para utilizar a biblioteca Chainer versão 3.

Fizemos um treino supervisionado de 40 épocas obtendo o mínimo de perda de erro com valor de 0.071 como podemos ver no resultado apresentado na figura 9. A perda de erro, ou *Loss*, é a diferença entre a saída esperada com o valor da saída do treinamento.

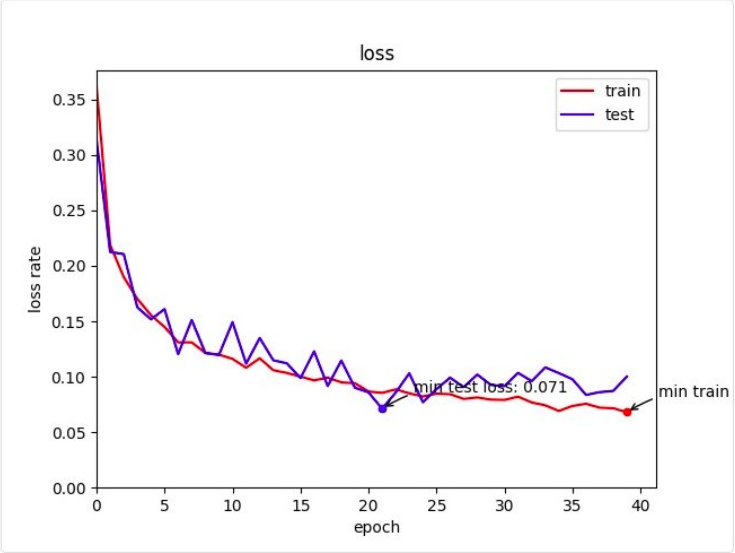


Figura 9 – Treinamento loss gráfico

5 ANÁLISE E RESULTADOS

Como mencionamos na seção 4.2 as análises dos resultados são realizadas por cada *frame* de um vídeo, então coletamos algumas amostras de frames para posterior diagnóstico. Como o modelo *Hyperface* reconhece mais informações do que apenas a pose da cabeça faremos também as observações do gênero e das marcações faciais. Analisaremos três datasets: vídeo da webcam, vídeo obtido do youtube e vídeo gravado na universidade de Salford.

5.1 ANÁLISE VÍDEO WEBCAM

As melhores detecções que obtivemos foram de imagens gravadas da própria webcam. Como essas imagens estão mais parecidas com as imagens do *dataset* utilizado no treinamento o resultado consequentemente iria ser melhor, porém isso foi concluído ao final de todos as outras análises. Nas figuras 10, 11, 12, 13 e 14 podemos ver que foi uma movimentação angular da cabeça da pessoa que começou olhando para o lado esquerdo da câmera e seguiu até o lado superior direito como podemos ver na Figura 14.

Nas Figuras 10 e 11, o algoritmo acertou tanto o gênero, o qual explicamos anteriormente na seção 4.2 que é indicado pela bolinha vermelha no canto superior direito, como a posição facial e os pontos faciais.

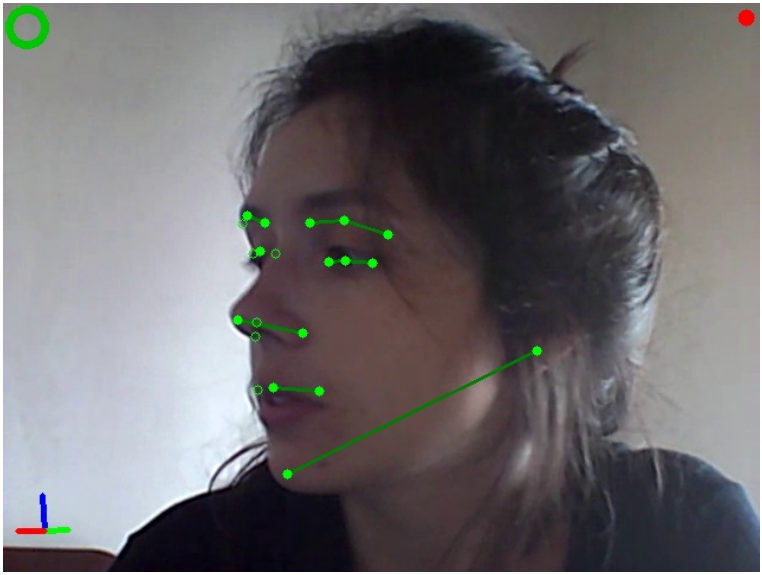


Figura 10 – Frame da Webcam 1

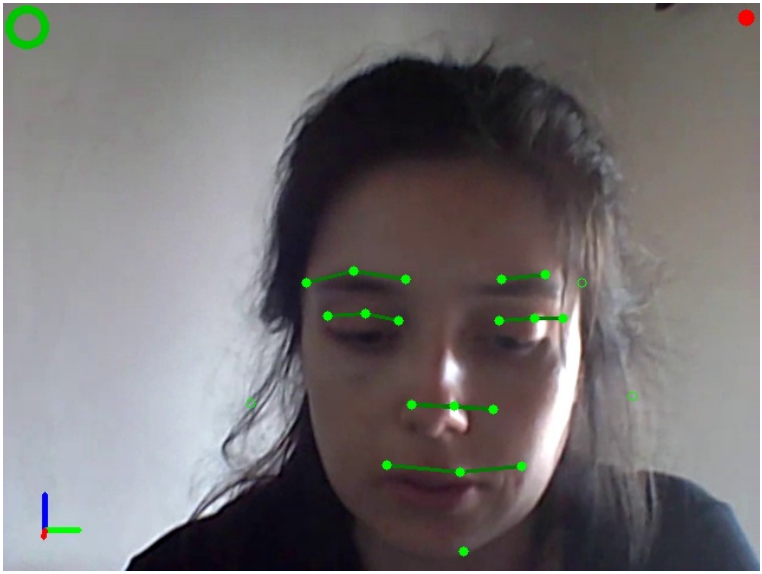


Figura 11 – Frame da Webcam 2

No entanto na figura 12 a rede errou os pontos faciais, isso se deu pelo fato do queixo ter ficado de fora da captura da câmera, porém o gênero e a posição facial continuaram corretas.

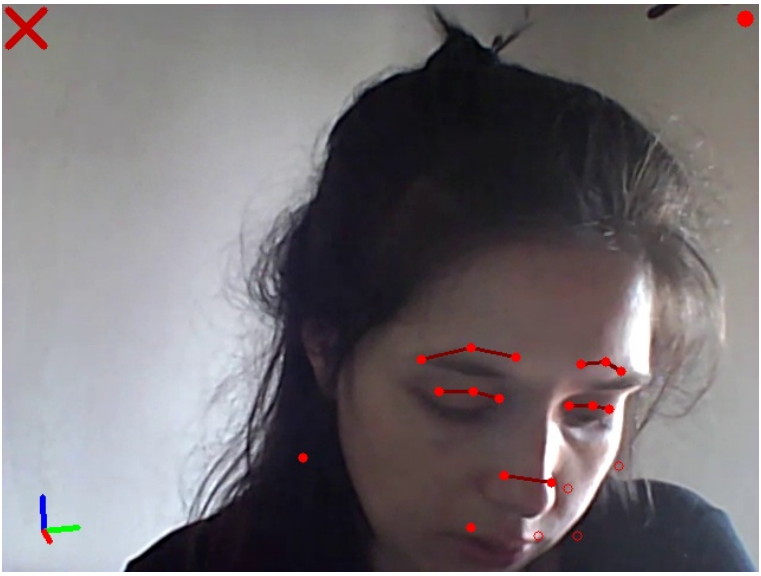


Figura 12 – Frame da Webcam 3

Na figura 13 assim como na 10 e 11 todas as 3 saídas da rede neural foram classificadas corretas, podemos ver bem o posicionamento diferente da triangulação do *roll, pitch* e *yaw* em cada uma das imagens.

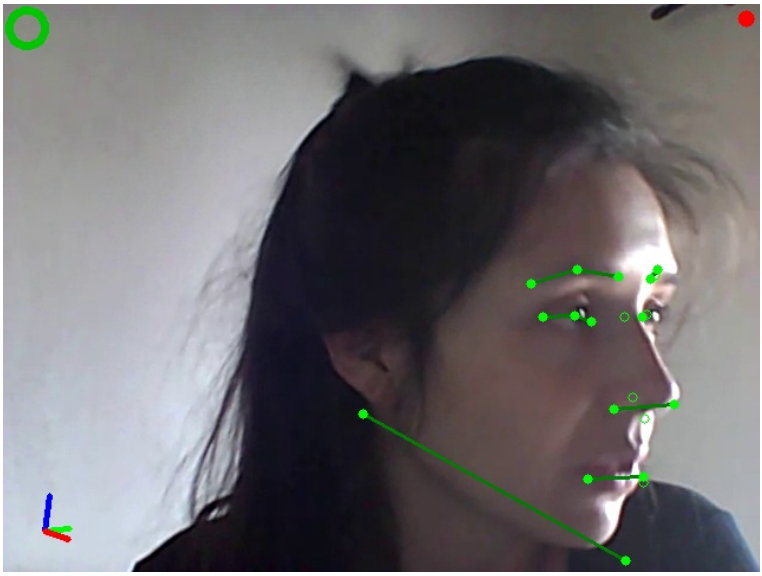


Figura 13 – Frame da Webcam 4

Completando essa análise, a figura 14 errou o gênero, podemos ver a bolinha azul indicando que trata-se do gênero masculino.

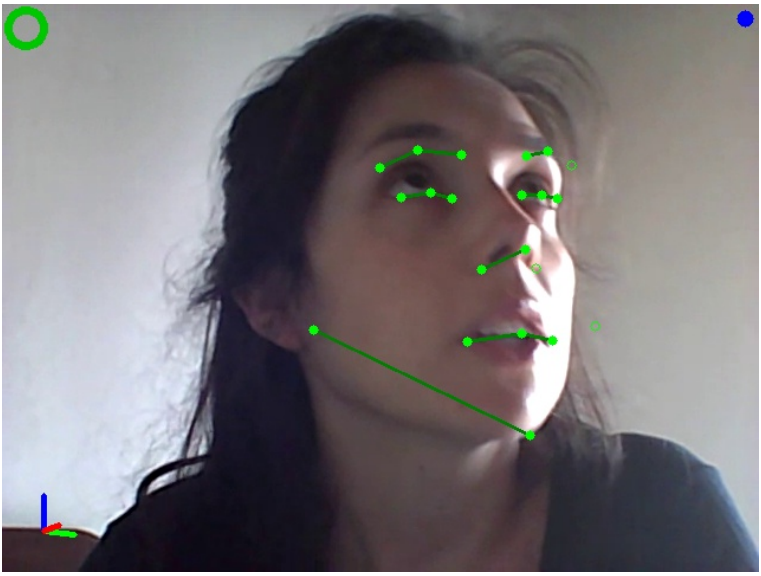


Figura 14 – Frame da Webcam 5

Para concluir, a análise da webcam obteve um resultado bastante satisfatório, além de não ter errado em nenhum momento o posicionamento facial, as indicações de gênero e os pontos faciais tiveram muitos acertos. Na tabela 1 podemos ver o resultado da análise por cada figura.

	Posicionamento da Cabeça	Gênero	Pontos Faciais
Figura 10	Acertou	Acertou	Acertou
Figura 11	Acertou	Acertou	Acertou
Figura 12	Acertou	Acertou	Errou
Figura 13	Acertou	Acertou	Acertou
Figura 14	Acertou	Errou	Acertou

Tabela 1 – Análise vídeo da webcam

5.2 ANÁLISE VÍDEO YOUTUBE

Utilizamos o trailer do filme "Os Vingadores: Guerra Infinita"para a análise de vídeo do youtube. Primeiro frame que vamos analisar é a Figura 15, nesta podemos ver que foram acertados as 3 saídas da Rede

Neural. Um fator importante de notar é que o código foi feito para identificar apenas um rosto por frame. Uma sugestão de melhoria para uma próxima versão é identificar todos os rostos da imagem.

Figura 15 – Frame do Youtube 1

Na Figura 16 a RNC errou todas as saídas. A posição facial, apesar de quase estar certa, é notável que o rosto do personagem está olhando mais lateralmente do que a indicação da triangulação.

Figura 16 – Frame do Youtube 2

As Figuras 17, 18, 19 e 20 serão analisadas juntas, o filme escolhido não foi por acaso, a ideia era ver como a rede neural ia se comportar analisando personagens diferentes de pessoas. Pelo resultado obtido, não tivemos algo satisfatório, com exceção da figura 18 que aparentemente teve a posição facial correta, porém acredito que tenha sido um acaso. Todas as outras figuras desse conjunto tiveram a análise errada. Para completar na Figura 20 a RNA identificou uma parte de rosto na mão do personagem.

Figura 17 – Frame do Youtube 3

Figura 18 – Frame do Youtube 4

Figura 19 – Frame do Youtube 5

Figura 20 – Frame do Youtube 6

As próximas análises das figuras 21, 22 e 23 foram corretas, porém como já sabemos a RNA foi treinada com imagens onde os rostos estão com mais zoom, ou seja, que estejam numa escala maior de aproximação da tela, então era de se esperar um resultado melhor.

Figura 21 – Frame do Youtube 7

Figura 22 – Frame do Youtube 8

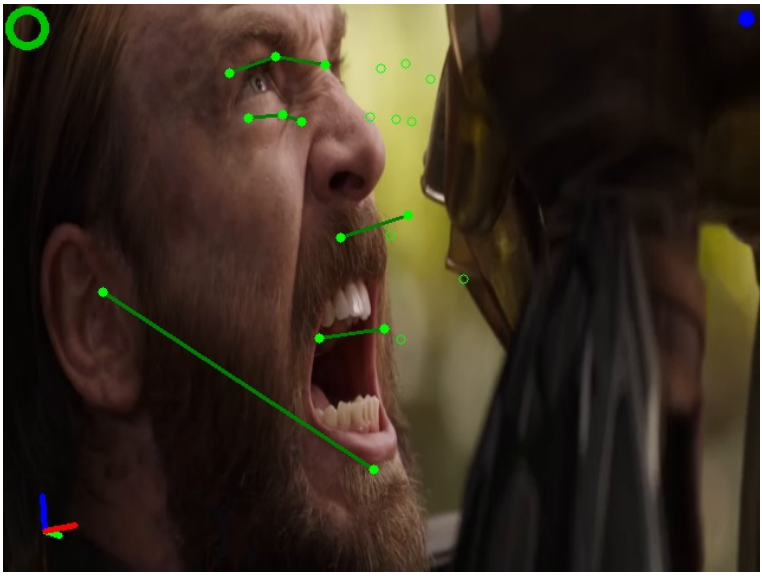


Figura 23 – Frame do Youtube 9

Apesar da rede neural ter errado alguns pontos, principalmente nas imagens em que os rostos estavam mais afastados e também quando apareciam personagens diferentes de humanos o resultado foi satisfatório. Se fizer um treinamento melhor e com um *dataset* com imagens diferente, principalmente utilizando imagens com rostos mais afastados acredito que o resultado seja muito melhor. Na tabela 2 podemos ver o resultado da análise por cada figura.

	Posicionamento da Cabeça	Gênero	Pontos Faciais
Figura 15	Acertou	Acertou	Acertou
Figura 16	Errou	Errou	Errou
Figura 17	Errou	Errou	Errou
Figura 18	Acertou	Errou	Errou
Figura 19	Errou	Errou	Errou
Figura 20	Errou	Errou	Errou
Figura 21	Acertou	Acertou	Acertou
Figura 22	Acertou	Acertou	Acertou
Figura 23	Acertou	Acertou	Acertou

Tabela 2 – Análise vídeo do youtube

5.3 ANÁLISE VÍDEO SALFORD

A idéia que deu origem ao trabalho era fazer a análise em um vídeo gravado na universidade de Salford. Posteriormente colocamos como objetivo principal a análise desse vídeo e infelizmente foi onde alcançamos os piores resultados. Um dos motivos foi pela qualidade do vídeo, pois como o zoom do vídeo estava muito afastado editamos para aproximá-lo, consequentemente com isso perdemos muita qualidade. O outro motivo que eu acredito que tenha prejudicado nos acertos do resultado foi o óculos utilizado pela pessoa que está sendo gravada.

Podemos ver os resultados nas figuras 24, 25, 26, 27 e 28. Todos as análises não foram satisfatórias, na figura 27 aparentemente a *Hyperface* acertou a movimentação do rosto, porém acredito que esteja analisando os pontos faciais errados, então acertou por acaso. Assim como na Figura 28, podemos ver que a RNA considerou a pessoa olhando bem de frente, o que não é verdade.

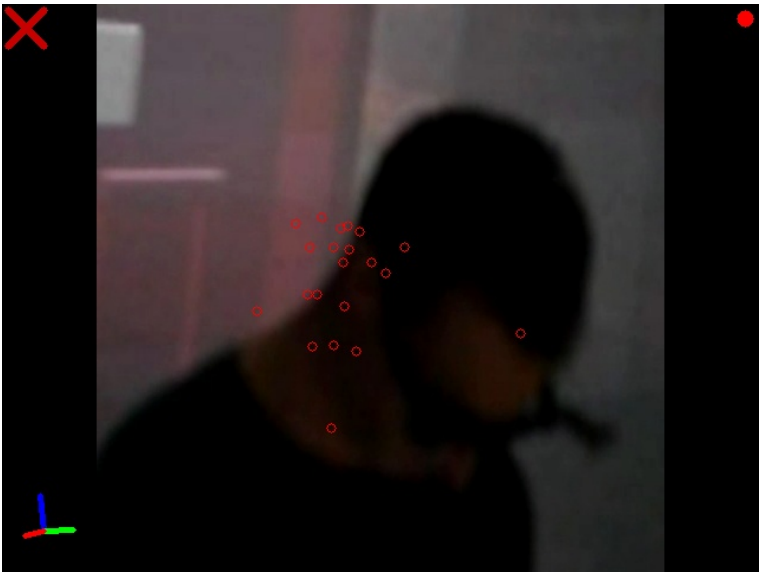


Figura 24 – Frame de Salford 1

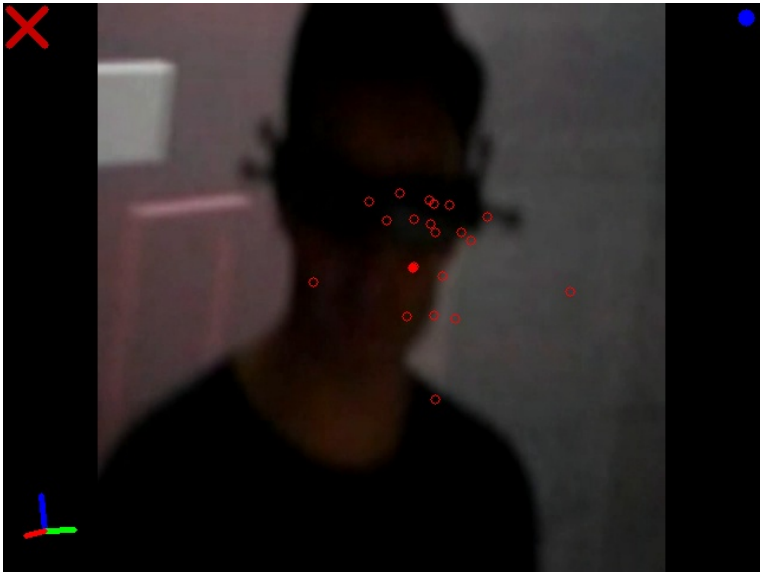


Figura 25 – Frame de Salford 2

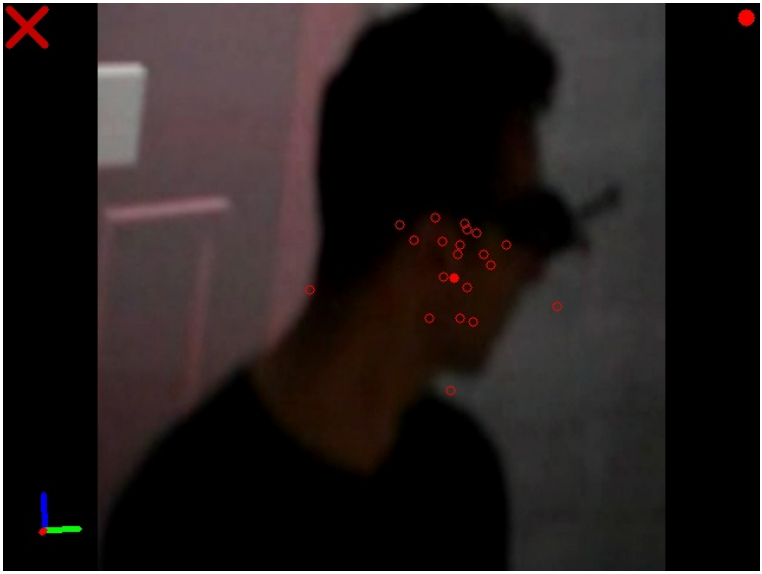


Figura 26 – Frame de Salford 3

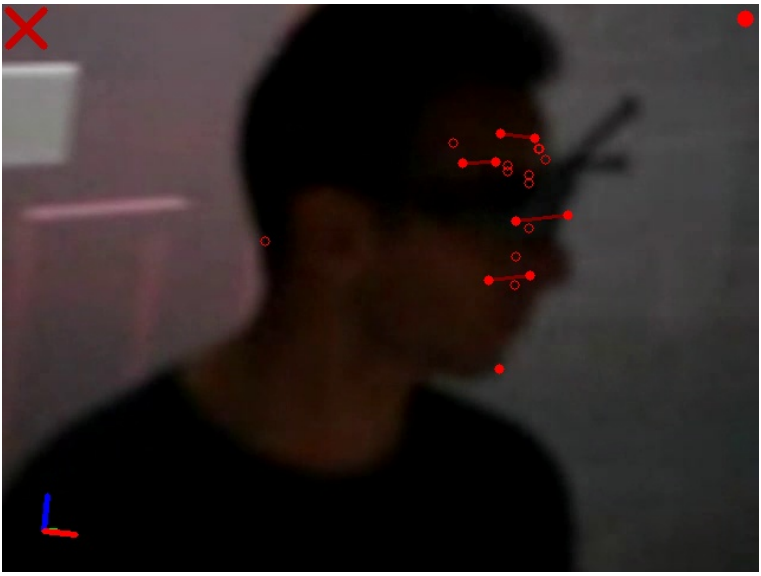


Figura 27 – Frame de Salford 4

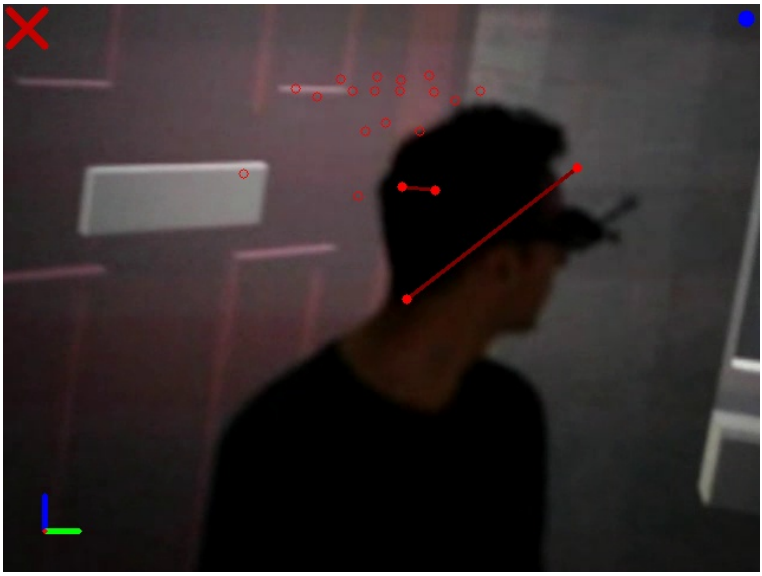


Figura 28 – Frame de Salford 5

Para concluir no vídeo de 1 minuto e 32 segundos o algoritmo não conseguiu acertar um frame que contenha todas as 3 saídas da Rede Neural corretas. Podemos ver o resultado da análise na Tabela 3.

	Posicionamento da Cabeça	Gênero	Pontos Faciais
Figura 24	Errou	Errou	Errou
Figura 25	Errou	Errou	Errou
Figura 26	Errou	Errou	Errou
Figura 27	Acertou	Errou	Errou
Figura 28	Errou	Errou	Errou

Tabela 3 – Análise vídeo de Salford

6 CONCLUSÃO

Inteligência Artificial é um assunto muito em alta hoje em dia. Fizemos uma pesquisa de redes neurais para analisar a movimentação corporal de uma pessoa restringindo a movimentação facial. Esse assunto pode ser aplicado nas áreas de carros autônomos, segurança e até deficiências motoras.

Apesar de não ter atingido o resultado esperado do nosso objetivo principal que era a análise do vídeo gravado na Universidade de Salford. Todos os objetivos específicos foram alcançados, pesquisamos alguns modelos de inteligência artificial e por fim propomos, treinamos e avaliamos um modelo de rede neural de reconhecimento de movimentação facial.

Acredito que conquistamos um resultado satisfatório no vídeo da webcam onde apresentamos a análise de cinco *frames* e conseguimos 100% de acerto no posicionamento da cabeça e também no vídeo do youtube onde foram analisados nove frames e obtivemos cinco *frames* corretos no mesmo quesito. Os resultados alcançados nos vídeos da webcam e youtube se deram pelo fato de possuírem qualidade superiores em comparação ao vídeo obtido da Salford e também estavam mais alinhados com o *dataset* utilizado para o treinamento.

Criando um dataset com os dados de Salford para realizar o treinamento com certeza conquistaríamos resultados melhores para a validação do vídeo. Entretanto como já vimos na seção 4.3.2.3 utilizamos o treinamento supervisionado que foi detalhado em 2.1 e a criação de um *dataset* tomaria muito tempo para ser concluído em um trabalho de conclusão de curso, pois teríamos que colocar as anotações faciais, o gênero e os valores de *Pitch, Yaw* e *Roll* para cada *frame*.

6.1 TRABALHOS FUTUROS

Para trabalhos futuros seria interessante fazer a modificação do código para aceitar mais de um rosto por vídeo e também fazer um treinamento com um dataset que tenha imagens de pessoas que estão distantes da câmera.

Fazendo um pré-processamento dos dados utilizando uma outra rede neural que identificasse um rosto, por exemplo a openface proposta por Baltrusaitis et al. (2018) e utilizando o resultado dela como entrada para esse trabalho obteríamos um resultado melhor.

Outra sugestão seria fazer todo o código utilizando a biblioteca *TensorFlow* ao invés de utilizar a *Chainer v3* que foi utilizada nesse projeto. Porque ao realizar o treinamento no computador local muitas vezes ocorreu exceção de limite de memória mesmo tendo 8gb de ram, então utilizamos um servidor da Amazon c3.x2large para conseguir treinar sem erro. Além disso a comunidade e a documentação do *TensorFlow* é muito mais completa e não tem tantas mudanças de uma versão para a outra como foi o caso do *Chainer* da versão 1 para a 3.

REFERÊNCIAS

BALTRUSAITIS, T. et al. Openface 2.0: Facial behavior analysis toolkit. In: *2018 13th IEEE International Conference on Automatic Face Gesture Recognition (FG 2018)*. [S.l.: s.n.], 2018. p. 59–66.

BENGIO, Y.; GOODFELLOW, I. J.; COURVILLE, A. Deep learning. *Nature*, v. 521, p. 436–444, 2015.

COPPIN, B. *Inteligência Artificial*. Rio de Janeiro, Brasil: LTC, 2013. 636 p.

DROUARD, V. et al. Head pose estimation via probabilistic high-dimensional regression. In: IEEE. *Image Processing (ICIP), 2015 IEEE International Conference on*. [S.l.], 2015. p. 4624–4628.

GIRSHICK, R. et al. Rich feature hierarchies for accurate object detection and semantic segmentation. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2014. p. 580–587. ISSN 1063-6919.

GOURIER D. HALL, J. L. C. N. Estimating face orientation from robust detection of salient facial features. *International Workshop on Visual Observation of Deictic Gestures*, 2004. <http://www-prima.inrialpes.fr/perso/Gourier/Faces/HPDatabase.html>.

GOURIER, N.; HALL, D.; CROWLEY, J. L. Estimating face orientation from robust detection of salient facial structures. In: *FG NET WORKSHOP ON VISUAL OBSERVATION OF DEICTIC GESTURES*. [S.l.: s.n.], 2004.

HAYKIN, S. *Redes Neurais*: princípios e prática. 2. ed. Porto Alegre, Brasil: Bookman, 2001. 900 p.

KOESTINGER PAUL WOHLHART, P. M. R. M.; BISCHOF, H. Annotated Facial Landmarks in the Wild: A Large-scale, Real-world Database for Facial Landmark Localization. In: *Proc. First IEEE International Workshop on Benchmarking Facial Image Analysis Technologies*. [S.l.: s.n.], 2011.

NAIR, V.; HINTON, G. E. Rectified linear units improve restricted boltzmann machines. In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. [S.l.: s.n.], 2010. p. 807–814.

NGUYEN, D. T.; LI, W.; OGUNBONA, P. O. Human detection from images and videos: a survey. pattern recognition. p. 51:148–175, 2016.

PATACCHIOLA M., . C. A. Head pose estimation in the wild using convolutional neural networks and adaptive gradient methods. pattern recognition. *arXiv preprint arXiv:1511.04031*, 2017. <http://dx.doi.org/10.1016/j.patcog.2017.06.009>.

RANJAN, R.; PATEL, V. M.; CHELLAPPA, R. Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition. *arXiv preprint arXiv:1603.01249*, 2016.

RICCI, E.; ODOBEZ, J. M. Learning large margin likelihoods for realtime head pose tracking. In: *2009 16th IEEE International Conference on Image Processing (ICIP)*. [S.l.: s.n.], 2009. p. 2593–2596. ISSN 1522-4880.

RUSSEL, S.; NORVIG, P. *Inteligência Artificial*. Rio de Janeiro, Brasil: ed. Elsevier, 2004. 1021 p.

SEEMANN, E.; NICKEL, K.; STIEFELHAGEN, R. Head pose estimation using stereo vision for human-robot interaction. In: *Sixth IEEE International Conference on Automatic Face and Gesture Recognition, 2004. Proceedings.* [S.l.: s.n.], 2004. p. 626–631.

TAIKU. *Deep Neural Network (DNN) which predicts face/non-face, landmarks, pose and gender simultaneously with Chainer*. nov 2017. <https://github.com/takiyu/hyperface>.

TURING, A. M. Computing machinery and intelligence. *Mind*, JSTOR, v. 59, n. 236, p. 433–460, 1950.

WANG, H. *Detection of Humans in Video Streams Using Convolutional Neural Networks*. 2017.

WU, Y. et al. Facial landmark detection with tweaked convolutional neural networks. *arXiv preprint arXiv:1511.04031*, 2016. First two authors are equal contributors / joint first authors. <https://www.openu.ac.il/home/hassner/projects/tcnn$_l$*andmarks*>.

ZHU, X.; RAMANAN, D. Face detection, pose estimation, and landmark localization in the wild. In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2012. p. 2879–2886. ISSN 1063-6919.

APÊNDICE A – Código Fonte

Arquivo use_on_video.py

```
import cv2
import numpy as np
import argparse
import chainer
import os

import config
import drawing
import log_initializer
import models
import gc
import time
import psutil

# logging
from logging import getLogger, DEBUG
log_initializer.setFmt()
log_initializer.setRootLevel(DEBUG)
logger = getLogger(__name__)

# Disable type check in chainer
os.environ["CHAINER_TYPE_CHECK"] = "0"

def _cvt_variable(v):
    # Convert from chainer variable
    if isinstance(v, chainer.variable.Variable):
        v = v.data
        if hasattr(v, 'get'):
            v = v.get()
    return v

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='HyperFace training script')
    parser.add_argument('--config', '-c', default='config.json',
                        help='Load config from given json file')
    parser.add_argument('--model', required=True, help='Trained model path')
    parser.add_argument('--input', required=True, help='Input video path')
    parser.add_argument('--output', required=True, help='Output video path')
    args = parser.parse_args()


```

```
42      # Create a VideoCapture object and read from input file
43      # If the input is the camera, pass 0 instead of the video file name
44      cap = cv2.VideoCapture(args.input)
45
46      # Check if file opened successfully
47      if (cap.isOpened() == False):
48          print("Error opening video stream or file")
49          cap.release()
50          cv2.destroyAllWindows()
51          exit(0)
52
53      logger.info('HyperFace Evaluation')
54
55      # Load config
56      config.load(args.config)
57
58      # Define a model
59      logger.info('Define a HyperFace model')
60      model = models.HyperFaceModel()
61      model.train = False
62      model.report = False
63      model.backward = False
64
65      # Initialize model
66      logger.info('Initialize a model using model "{}"'.format(args.model))
67      chainer.serializers.load_npz(args.model, model)
68
69      # Setup GPU
70      if config.gpu >= 0:
71          chainer.cuda.check_cuda_available()
72          chainer.cuda.get_device(config.gpu).use()
73          model.to_gpu()
74          xp = chainer.cuda.cupy
75      else:
76          xp = np
77
78      success,image = cap.read()
79      count = 0
80      success = True
81      frame_width = int(cap.get(3))
82      frame_height = int(cap.get(4))
83      print(frame_width)
84      print(frame_height)
```

```
85
86      fourcc = cv2.VideoWriter_fourcc(*'XVID')
87      print(fourcc)
88      out = cv2.VideoWriter(args.output,fourcc, 30.0, (frame_width,frame_height))
89      while success:
90          success,image = cap.read()
91
92          print('Read a new frame: ', success)
93          count += 1
94          if (success):
95              # Load image file
96              img = image
97              img2 = image
98              if img is None or img.size == 0 or img.shape[0] == 0 or img.shape[1] ==
99                  print('Failed to load image')
100                 logger.error('Failed to load')
101                 exit(0)
102             img = img.astype(np.float32) / 255.0  # [0:1]
103             img = cv2.resize(img, models.IMG_SIZE)
104             img = cv2.normalize(img, None, -0.5, 0.5, cv2.NORM_MINMAX)
105             img = np.transpose(img, (2, 0, 1))
106
107             # Create single batch
108             imgs = xp.asarray([img])
109             with chainer.no_backprop_mode():
110                 x = chainer.Variable(imgs)
111
112                 # Forward
113                 logger.info('Forward the network')
114                 y = model(x)
115
116                 # Chainer.Variable -> np.ndarray
117                 imgs = _cvt_variable(y['img'])
118                 detections = _cvt_variable(y['detection'])
119                 landmarks = _cvt_variable(y['landmark'])
120                 visibilitys = _cvt_variable(y['visibility'])
121                 poses = _cvt_variable(y['pose'])
122                 genders = _cvt_variable(y['gender'])
123
124                 # Use first data in one batch
125                 img = imgs[0]
126                 detection = detections[0]
127                 landmark = landmarks[0]
```

```
128                     visibility = visibilitys[0]
129                     pose = poses[0]
130                     gender = genders[0]
131
132                     img = np.transpose(img, (1, 2, 0))
133                     img = img.copy()
134                     img += 0.5  # [-0.5:0.5] -> [0:1]
135                     print(detection)
136                     detection = (detection > 0.5)
137                     gender = (gender > 0.5)
138
139                     # Draw results
140                     drawing.draw_detection(img, detection)
141                     landmark_color = (0, 255, 0) if detection == 1 else (0, 0, 255)
142                     drawing.draw_landmark(img, landmark, visibility, landmark_color, 0.5)
143                     # Descobrir as outras poses o que querem dizer em relacao a cabeca.
144                     # pose[0] -> cabeca virada pros lados roll (rolando)
145                     # pose[1] -> cabeca virada pra cima ou baixo pitch
146                     # pose[2] -> cabeca olhando pra qual lado yaw
147                     limite = 0.1
148                     if (pose[1]<-limite):
149                         print ('Cabeca em estado para Baixo')
150                     elif(pose[1]>limite):
151                         print('Cabeca em estado para Cima')
152                     elif(pose[1]>=-limite and pose[2]<=limite):
153                         print('Cabeca em estado Reto')
154                     if (pose[2]<-limite):
155                         print ('Olhando para a esquerda')
156                     elif (pose[2]>limite):
157                         print('Olhando para a direita')
158                     elif (pose[2]>=-limite and pose[2]<=limite):
159                         print('Olhando para frente')
160                     print(pose)
161                     drawing.draw_pose(img, pose)
162                     drawing.draw_gender(img, gender)
163
164                     img2 = cv2.resize(img2, (640, 480))
165                     drawing.draw_detection(img2, detection)
166                     drawing.draw_landmark(img2, landmark, visibility, landmark_color, 0.5)
167                     drawing.draw_pose(img2, pose)
168                     drawing.draw_gender(img2, gender)
169                     out.write(img2)
170                     # Show image
```

```
171                 #logger.info('Show the result image')
172                 #cv2.imshow('result', img)
173                 #cv2.waitKey(0)
174                 cv2.imwrite("frames/frame%d.jpg" % count, img2)     # save frame a
175                 img2 = None
176                 img = None
177                 gc.collect()
178 
179     # When everything done, release the video capture object
180     cap.release()
181     out.release()
182 
183     # Closes all the frames
184     cv2.destroyAllWindows()
```

Arquivo train.py

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  
4  import chainer
5  from chainer import training
6  from chainer import iterators
7  from chainer.training import extensions
8  from chainer.links.caffe import CaffeFunction
9  
10 import argparse
11 import os
12 
13 import log_initializer
14 
15 import config
16 import datasets
17 from extensions import ImgViewerExtention
18 from extensions import SequentialEvaluator
19 from imgviewer_conversions import face_img_func, weights_img_func
20 from imgviewer_conversions import lossgraph_entry_func, lossgraph_img_func
21 import models
22 from models import copy_layers
23 
24 # logging
25 from logging import getLogger, DEBUG
26 log_initializer.setFmt()
```

```
27  log_initializer.setRootLevel(DEBUG)
28  logger = getLogger(__name__)
29
30  # Disable type check in chainer
31  os.environ["CHAINER_TYPE_CHECK"] = "0"
32
33
34  if __name__ == '__main__':
35      # Argument
36      parser = argparse.ArgumentParser(description='HyperFace training script')
37      parser.add_argument('--config', '-c', default='config.json',
38                          help='Load config from given json file')
39      parser.add_argument('--resume', '-r', default='',
40                          help='Initialize the trainer from given file')
41      parser.add_argument('--pretrain', action='store_true',
42                          help='Flag for pretraining mode')
43      parser.add_argument('--pretrainedmodel', default='',
44                          help='Initialize the model using pretrained')
45      args = parser.parse_args()
46
47      logger.info('HyperFace Training (Mode: {}'
48                  .format('Pretrain' if args.pretrain else 'Main'))
49
50      # Load config
51      config.load(args.config)
52
53      # Setup AFLW dataset
54      train, test = datasets.setup_aflw(config.aflw_cache_path,
55                                        config.aflw_sqlite_path,
56                                        config.aflw_imgdir_path,
57                                        config.aflw_test_rate)
58
59      # Define a model
60      if args.pretrain:
61          logger.info('Define a R-CNN_Face model')
62          model = models.RCNNFaceModel()
63      else:
64          logger.info('Define a HyperFace model')
65          model = models.HyperFaceModel()
66
67      # Initialize model
68      if not args.resume:
69          if args.pretrain and config.alexnet_caffemodel_path:
```

```
70              # Initialize using caffemodel
71              logger.info('Overwrite conv layers using caffemodel "{}"'
72                          .format(config.alexnet_caffemodel_path))
73              caffe_model = CaffeFunction(config.alexnet_caffemodel_path)
74              copy_layers(caffe_model, model)
75          elif not args.pretrain and args.pretrainedmodel:
76              # Initialize using pretrained model
77              logger.info('Overwrite conv layers using pretraindmodel "{}"'
78                          .format(args.pretrainedmodel))
79              pre_model = models.RCNNFaceModel()
80              chainer.serializers.load_npz(args.pretrainedmodel, pre_model)
81              copy_layers(pre_model, model)
82
83      # Setup GPU
84      if config.gpu >= 0:
85          chainer.cuda.check_cuda_available()
86          chainer.cuda.get_device(config.gpu).use()
87          model.to_gpu()
88
89      # Setup an optimizer
90      logger.info('Setup an optimizer')
91      optimizer = chainer.optimizers.MomentumSGD(lr=0.01, momentum=0.9)
92      optimizer.setup(model)
93
94      # Setup iterators
95      # (test_iter needs repeat because of SequentialEvaluator())
96      logger.info('Setup train and test iterators (n_loaders: {}, {})'
97                  .format(config.n_loaders_train, config.n_loaders_test))
98      train_iter = iterators.MultiprocessIterator(train, config.batchsize, True,
99                                                  True, config.n_loaders_train)
100     test_iter = iterators.MultiprocessIterator(test, config.batchsize, True,
101                                                True, config.n_loaders_test)
102
103     # Setup a updater
104     logger.info('Setup an updater (GPU: {})'.format(config.gpu))
105     updater = training.StandardUpdater(
106         train_iter, optimizer, device=config.gpu)
107     # Setup a trainer
108     outdir = config.outdir_pretrain if args.pretrain else config.outdir
109     logger.info('Setup a trainer (output directory: {})'.format(outdir))
110     trainer = training.Trainer(updater, (config.n_epoch, 'epoch'), out=outdir)
111
112     # Evaluation model with shared parameters
```

```
113     logger.info('Create a copy of the model for evaluation')
114     eval_model = model.copy()
115     eval_model.train = False
116 
117     # Extension intervals
118     n_iteration = max(len(train) // config.batchsize, 1)
119     test_interval = (max(len(train) // len(test), 1), 'iteration')
120     save_interval = (5, 'epoch')
121     log_interval = (max(n_iteration // 1, 1), 'iteration')
122     progressbar_interval = 3
123     imgview_face_interval = (5, 'iteration')
124     imgview_weight_interval = (1, 'epoch')
125     logger.info('Test interval : {}'.format(test_interval))
126     logger.info('Save interval : {}'.format(save_interval))
127     logger.info('Log interval :  {}'.format(log_interval))
128     logger.info('ProgressBar interval :  {}'.format(progressbar_interval))
129     logger.info('ImgView face interval :   {}'.format(imgview_face_interval))
130     logger.info('ImgView weight interval : {}'.format(imgview_weight_interval))
131 
132     # Extensions
133     trainer.extend(extensions.dump_graph('main/loss'), trigger=save_interval)
134     trainer.extend(extensions.snapshot(
135         filename='snapshot_epoch_{.updater.epoch}'), trigger=save_interval)
136     trainer.extend(extensions.snapshot_object(
137         model, 'model_epoch_{.updater.epoch}'), trigger=save_interval)
138     trainer.extend(extensions.LogReport(trigger=log_interval))
139     trainer.extend(extensions.PrintReport(
140         ['epoch', 'iteration', 'main/loss', 'validation/main/loss']),
141         trigger=log_interval)
142     trainer.extend(extensions.ProgressBar(
143         update_interval=progressbar_interval))
144 
145     # My extensions
146     # Sequential Evaluator
147     trainer.extend(
148         SequentialEvaluator(test_iter, eval_model, device=config.gpu),
149         trigger=test_interval)  # Sequential evaluation for imgviewer in test
150     # Image Viewer for Faces
151     trainer.extend(ImgViewerExtention(
152         ['main/predict', 'main/teacher', 'validation/main/predict',
153          'validation/main/teacher'], n_imgs=[20, 20, 10, 10],
154         port=config.port_face, image_func=face_img_func),
155         trigger=imgview_face_interval)
```

```
156     # Image Viewer for weights
157     trainer.extend(ImgViewerExtention(
158         ['main/conv1_w', 'main/conv2_w', 'main/conv3_w', 'main/conv4_w',
159          'main/conv5_w', ], n_imgs=[96, 0, 0, 0, 0],
160         port=config.port_weight, image_func=weights_img_func),
161         trigger=imgview_weight_interval)
162     # Image Viewer for loss graph
163     trainer.extend(ImgViewerExtention(
164         ['lossgraph'], n_imgs=[1] if args.pretrain else [1 + 5],
165         port=config.port_lossgraph, entry_func=lossgraph_entry_func,
166         image_func=lossgraph_img_func), trigger=log_interval)
167
168
169
170     # Resume
171     if args.resume:
172         logger.info('Resume from "{}"'.format(args.resume))
173         chainer.serializers.load_npz(args.resume, trainer)
174
175     # Run
176     logger.info('Start training')
177     trainer.run()
```

Arquivo models.py

```
1   # -*- coding: utf-8 -*-
2
3   import chainer
4   import chainer.functions as F
5   import chainer.links as L
6
7   # logging
8   from logging import getLogger, NullHandler
9   logger = getLogger(__name__)
10  logger.addHandler(NullHandler())
11
12  # Constant variables
13  N_LANDMARK = 21
14  #IMG_SIZE = (227, 227)
15  IMG_SIZE = (227, 227)
16
17  def _disconnect(x):
18      variableX = chainer.Variable(x.data)
```

```
19      with chainer.no_backprop_mode():
20          return chainer.Variable(x.data)
21
22
23  def copy_layers(src_model, dst_model,
24                  names=['conv1', 'conv2', 'conv3', 'conv4', 'conv5']):
25      for name in names:
26          for s, d in zip(src_model[name].params(), dst_model[name].params()):
27              d.data = s.data
28
29
30  class HyperFaceModel(chainer.Chain):
31
32      def __init__(self, loss_weights=(1.0, 100.0, 20.0, 5.0, 0.3)):
33          super(HyperFaceModel, self).__init__(
34              conv1=L.Convolution2D(3, 96, 11, stride=4, pad=0),
35              conv1a=L.Convolution2D(96, 256, 4, stride=4, pad=0),
36              conv2=L.Convolution2D(96, 256, 5, stride=1, pad=2),
37              conv3=L.Convolution2D(256, 384, 3, stride=1, pad=1),
38              conv3a=L.Convolution2D(384, 256, 2, stride=2, pad=0),
39              conv4=L.Convolution2D(384, 384, 3, stride=1, pad=1),
40              conv5=L.Convolution2D(384, 256, 3, stride=1, pad=1),
41              conv_all=L.Convolution2D(768, 192, 1, stride=1, pad=0),
42              fc_full=L.Linear(6 * 6 * 192, 3072),
43              fc_detection1=L.Linear(3072, 512),
44              fc_detection2=L.Linear(512, 2),
45              fc_landmarks1=L.Linear(3072, 512),
46              fc_landmarks2=L.Linear(512, 42),
47              fc_visibility1=L.Linear(3072, 512),
48              fc_visibility2=L.Linear(512, 21),
49              fc_pose1=L.Linear(3072, 512),
50              fc_pose2=L.Linear(512, 3),
51              fc_gender1=L.Linear(3072, 512),
52              fc_gender2=L.Linear(512, 2),
53          )
54          self.train = True
55          self.report = True
56          self.backward = True
57          assert(len(loss_weights) == 5)
58          self.loss_weights = loss_weights
59
60      def __call__(self, x_img, t_detection=None, t_landmark=None,
61                   t_visibility=None, t_pose=None, t_gender=None,
```

```
62                  m_landmark=None, m_visibility=None, m_pose=None):
63          # Alexnet
64          h = F.relu(self.conv1(x_img))  # conv1
65          h = F.max_pooling_2d(h, 3, stride=2, pad=0)  # max1
66          h = F.local_response_normalization(h)  # norm1
67          h1 = F.relu(self.conv1a(h))  # conv1a
68          h = F.relu(self.conv2(h))  # conv2
69          h = F.max_pooling_2d(h, 3, stride=2, pad=0)  # max2
70          h = F.local_response_normalization(h)  # norm2
71          h = F.relu(self.conv3(h))  # conv3
72          h2 = F.relu(self.conv3a(h))  # conv3a
73          h = F.relu(self.conv4(h))  # conv4
74          h = F.relu(self.conv5(h))  # conv5
75          h = F.max_pooling_2d(h, 3, stride=2, pad=0)  # pool5
76
77          h = F.concat((h1, h2, h))
78
79          # Fusion CNN
80          h = F.relu(self.conv_all(h))  # conv_all
81          h = F.relu(self.fc_full(h))  # fc_full
82          with chainer.using_config('train', self.train):
83              h = F.dropout(h)
84          h_detection = F.relu(self.fc_detection1(h))
85          with chainer.using_config('train', self.train):
86              h_detection = F.dropout(h_detection)
87          h_detection = self.fc_detection2(h_detection)
88          h_landmark = F.relu(self.fc_landmarks1(h))
89          with chainer.using_config('train', self.train):
90              h_landmark = F.dropout(h_landmark)
91          h_landmark = self.fc_landmarks2(h_landmark)
92          h_visibility = F.relu(self.fc_visibility1(h))
93          with chainer.using_config('train', self.train):
94              h_visibility = F.dropout(h_visibility)
95          h_visibility = self.fc_visibility2(h_visibility)
96          h_pose = F.relu(self.fc_pose1(h))
97          with chainer.using_config('train', self.train):
98              h_pose = F.dropout(h_pose)
99          h_pose = self.fc_pose2(h_pose)
100         h_gender = F.relu(self.fc_gender1(h))
101         with chainer.using_config('train', self.train):
102             h_gender = F.dropout(h_gender)
103         h_gender = self.fc_gender2(h_gender)
104
```

```
105         # Mask and Loss
106         if self.backward:
107             # Landmark masking with visibility
108             m_landmark_ew = F.stack((t_visibility, t_visibility), axis=2)
109             m_landmark_ew = F.reshape(m_landmark_ew, (-1, N_LANDMARK * 2))
110 
111             # Masking
112             h_landmark *= _disconnect(m_landmark)
113             t_landmark *= _disconnect(m_landmark)
114             h_landmark *= _disconnect(m_landmark_ew)
115             t_landmark *= _disconnect(m_landmark_ew)
116             h_visibility *= _disconnect(m_visibility)
117             t_visibility *= _disconnect(m_visibility)
118             h_pose *= _disconnect(m_pose)
119             t_pose *= _disconnect(m_pose)
120 
121             # Loss
122             loss_detection = F.softmax_cross_entropy(h_detection, t_detection)
123             loss_landmark = F.mean_squared_error(h_landmark, t_landmark)
124             loss_visibility = F.mean_squared_error(h_visibility, t_visibility)
125             loss_pose = F.mean_squared_error(h_pose, t_pose)
126             loss_gender = F.softmax_cross_entropy(h_gender, t_gender)
127 
128             # Loss scaling
129             loss_detection *= self.loss_weights[0]
130             loss_landmark *= self.loss_weights[1]
131             loss_visibility *= self.loss_weights[2]
132             loss_pose *= self.loss_weights[3]
133             loss_gender *= self.loss_weights[4]
134 
135             loss = (loss_detection + loss_landmark + loss_visibility +
136                     loss_pose + loss_gender)
137 
138         # Prediction (the same shape as t_**, and [0:1])
139         h_detection = F.softmax(h_detection)[:, 1] # ([[y, n]] -> [d])
140         h_gender = F.softmax(h_gender)[:, 1] # ([[m, f]] -> [g])
141 
142         if self.report:
143             if self.backward:
144                 # Report losses
145                 chainer.report({'loss': loss,
146                                 'loss_detection': loss_detection,
147                                 'loss_landmark': loss_landmark,
```

```
148                                 'loss_visibility': loss_visibility,
149                                 'loss_pose': loss_pose,
150                                 'loss_gender': loss_gender}, self)
151
152             # Report results
153             predict_data = {'img': x_img, 'detection': h_detection,
154                             'landmark': h_landmark, 'visibility': h_visibility,
155                             'pose': h_pose, 'gender': h_gender}
156             teacher_data = {'img': x_img, 'detection': t_detection,
157                             'landmark': t_landmark, 'visibility': t_visibility,
158                             'pose': t_pose, 'gender': t_gender}
159             chainer.report({'predict': predict_data}, self)
160             chainer.report({'teacher': teacher_data}, self)
161
162             # Report layer weights
163             chainer.report({'conv1_w': {'weights': self.conv1.W},
164                             'conv2_w': {'weights': self.conv2.W},
165                             'conv3_w': {'weights': self.conv3.W},
166                             'conv4_w': {'weights': self.conv4.W},
167                             'conv5_w': {'weights': self.conv5.W}}, self)
168
169         if self.backward:
170             return loss
171         else:
172             return {'img': x_img, 'detection': h_detection,
173                     'landmark': h_landmark, 'visibility': h_visibility,
174                     'pose': h_pose, 'gender': h_gender}
175
176
177 class RCNNFaceModel(chainer.Chain):
178
179     def __init__(self):
180         super(RCNNFaceModel, self).__init__(
181             conv1=L.Convolution2D(3, 96, 11, stride=4, pad=0),
182             conv2=L.Convolution2D(96, 256, 5, stride=1, pad=2),
183             conv3=L.Convolution2D(256, 384, 3, stride=1, pad=1),
184             conv4=L.Convolution2D(384, 384, 3, stride=1, pad=1),
185             conv5=L.Convolution2D(384, 256, 3, stride=1, pad=1),
186             fc6=L.Linear(6 * 6 * 256, 4096),
187             fc7=L.Linear(4096, 512),
188             fc8=L.Linear(512, 2),
189         )
190         self.train = True
```

```
191
192     def __call__(self, x_img, t_detection, **others):
193         # Alexnet
194         h = F.relu(self.conv1(x_img))  # conv1
195         h = F.max_pooling_2d(h, 3, stride=2, pad=0)  # max1
196         h = F.local_response_normalization(h)  # norm1
197         h = F.relu(self.conv2(h))  # conv2
198         h = F.max_pooling_2d(h, 3, stride=2, pad=0)  # max2
199         h = F.local_response_normalization(h)  # norm2
200         h = F.relu(self.conv3(h))  # conv3
201         h = F.relu(self.conv4(h))  # conv4
202         h = F.relu(self.conv5(h))  # conv5
203         h = F.max_pooling_2d(h, 3, stride=2, pad=0)  # pool5
204         with chainer.using_config('train', self.train):
205                 h = F.dropout(F.relu(self.fc6(h)))  # fc6
206                 h = F.dropout(F.relu(self.fc7(h)))  # fc7
207         h_detection = self.fc8(h)  # fc8
208
209         # Loss
210         loss = F.softmax_cross_entropy(h_detection, t_detection)
211
212         chainer.report({'loss': loss}, self)
213
214         # Prediction
215         h_detection = F.argmax(h_detection, axis=1)
216
217         # Report results
218         predict_data = {'img': x_img, 'detection': h_detection}
219         teacher_data = {'img': x_img, 'detection': t_detection}
220         chainer.report({'predict': predict_data}, self)
221         chainer.report({'teacher': teacher_data}, self)
222
223         # Report layer weights
224         chainer.report({'conv1_w': {'weights': self.conv1.W},
225                         'conv2_w': {'weights': self.conv2.W},
226                         'conv3_w': {'weights': self.conv3.W},
227                         'conv4_w': {'weights': self.conv4.W},
228                         'conv5_w': {'weights': self.conv5.W}}, self)
229
230         return loss
```

Arquivo hyperface.py

```
# -*- coding: utf-8 -*-
import chainer

import cv2
import numpy as np
import six

import common
import datasets
# import drawing
import models

# logging
from logging import getLogger, NullHandler
logger = getLogger(__name__)
logger.addHandler(NullHandler())


def _cvt_variable(v):
    # Convert from chainer variable
    if isinstance(v, chainer.variable.Variable):
        v = v.data
        if hasattr(v, 'get'):
            v = v.get()
    return v


def _forward_with_rects(model, img_org, rects, batchsize, xp):
    # Crop and normalize
    cropped_imgs = list()
    for x, y, w, h in rects:
        img = img_org[int(y):int(y + h + 1), int(x):int(x + w + 1), :]
        img = cv2.resize(img, models.IMG_SIZE)
        img = cv2.normalize(img, None, -0.5, 0.5, cv2.NORM_MINMAX)
        img = np.transpose(img, (2, 0, 1))
        cropped_imgs.append(img)

    detections = list()
    landmarks = list()
    visibilities = list()
    poses = list()
    genders = list()
```

```
43
44      # Forward each batch
45      for i in six.moves.xrange(0, len(cropped_imgs), batchsize):
46          # Create batch
47          batch = xp.asarray(cropped_imgs[i:i + batchsize], dtype=np.float32)
48          x = chainer.Variable(batch, volatile=True)
49          # Forward
50          y = model(x)
51          # Chainer.Variable -> np.ndarray
52          detections.extend(_cvt_variable(y['detection']))
53          landmarks.extend(_cvt_variable(y['landmark']))
54          visibilities.extend(_cvt_variable(y['visibility']))
55          poses.extend(_cvt_variable(y['pose']))
56          genders.extend(_cvt_variable(y['gender']))
57
58      # Denormalize landmarks
59      for i, (x, y, w, h) in enumerate(rects):
60          landmarks[i] = landmarks[i].reshape(models.N_LANDMARK, 2)  # (21, 2)
61          landmark_offset = np.array([x + w / 2, y + h / 2], dtype=np.float32)
62          landmark_denom = np.array([w, h], dtype=np.float32)
63          landmarks[i] = landmarks[i] * landmark_denom + landmark_offset
64
65      return detections, landmarks, visibilities, poses, genders
66
67
68  def _proposal_region(pts, pts_mask, img_rect, landmark_pad_rate=0.1):
69      # TODO Improve for rotated rectangles
70      # AFLW Template
71      tpl_pts = datasets.aflw_template_landmark()
72      tpl_rect = cv2.boundingRect(tpl_pts)
73      # Mask points
74      mask_idxs = np.where(pts_mask > 0.5)
75      masked_pts = pts[mask_idxs]
76      masked_tpl_pts = tpl_pts[mask_idxs]
77      if masked_pts.shape[0] < 4 or masked_tpl_pts.shape[0] < 4:
78          return (0, 0, 0, 0)
79
80
81      # Homography matrix
82      H, _ = cv2.findHomography(masked_tpl_pts, masked_pts, method=cv2.RANSAC)
83      if H is None:
84          return (0, 0, 0, 0)
85      # Apply to rect
```

```
86      x1, y1 = tpl_rect[0], tpl_rect[1]
87      x2, y2 = tpl_rect[2] + x1, tpl_rect[3] + y1
88      rect_pts = np.array([[x1, y1, 1.0],
89                           [x1, y2, 1.0],
90                           [x2, y1, 1.0],
91                           [x2, y2, 1.0]], dtype=np.float32)
92      rect_pts = H.dot(rect_pts.T).T
93      rect_pts = rect_pts[:, 0:2] / rect_pts[:, 2].reshape(4, 1)
94      # Convert points to rectangle
95      min_xy = np.min(rect_pts, axis=0)
96      max_xy = np.max(rect_pts, axis=0)
97      rect = (min_xy[0], min_xy[1], max_xy[0] - min_xy[0], max_xy[1] - min_xy[1])
98
99      # landmark's rect
100     pts_rect = cv2.boundingRect(masked_pts)
101     # padding
102     x, y, w, h = pts_rect
103     pad_w = w * landmark_pad_rate
104     pad_h = h * landmark_pad_rate
105     pts_rect = (x - pad_w / 2.0, y - pad_h / 2.0, w + pad_w, h + pad_h)
106
107     # union with landmark's rect
108     rect = common.rect_or(rect, pts_rect)
109
110     # intersect with img_rect
111     rect = common.rect_and(rect, img_rect)
112     return rect
113
114
115 def _bounding_region(pts, pts_mask):
116     masked_pts = pts[np.where(pts_mask > 0.5)]
117     if masked_pts.shape[0] == 0:
118         return (0, 0, 0, 0)
119     return cv2.boundingRect(masked_pts)
120
121
122 class HyperFace(object):
123     def __init__(self, model_path, gpu=-1, batchsize=32):
124         # Define a model
125         logger.info('Define a HyperFace model using {}'.format(model_path))
126         model = models.HyperFaceModel()
127         chainer.serializers.load_npz(model_path, model)
128         model.train = False
```

```
129         model.report = False
130         model.backward = False
131 
132         # GPU
133         if gpu >= 0:
134             chainer.cuda.check_cuda_available()
135             chainer.cuda.get_device(gpu).use()
136             model.to_gpu()
137             xp = chainer.cuda.cupy
138         else:
139             xp = np
140 
141         # Variables
142         self.model = model
143         self.xp = xp
144         self.batchsize = batchsize
145 
146     def __call__(self, img):
147         # ========== Iterative Region Proposals (IRP) ==========
148         img_rect = (0, 0, img.shape[1], img.shape[0])
149         detections, landmarks, visibilities = None, None, None
150         for stage_cnt in six.moves.xrange(3):
151             if stage_cnt == 0:
152                 # Selective search, crop and normalize
153                 ssrects = common.selective_search_dlib(img,
154                                                        max_img_size=(999, 999),
155                                                        kvals=(50, 200, 2),
156                                                        min_size=1500,
157                                                        check=False,
158                                                        debug_window=False)
159             # Forward
160             detections, landmarks, visibilities, poses, genders = \
161                 _forward_with_rects(self.model, img, ssrects, self.batchsize,
162                                     self.xp)
163 
164             # Update ssrects using landmarks
165             new_ssrects = list()
166             for i in six.moves.xrange(len(ssrects)):
167                 if detections[i] > 0.25:  # TODO configure
168                     new_ssrect = _proposal_region(landmarks[i],
169                                                   visibilities[i], img_rect)
170                     if new_ssrect[2] > 0 and new_ssrect[3] > 0:
171                         new_ssrects.append(new_ssrect)
```

```
172             ssrects = new_ssrects
173
174         # [DEBUG] Draw IRP rectabgles
175 #         for rect in ssrects:
176 #             drawing._draw_rect(img, rect, (0, 1, 0))
177
178         # Extract detected entries
179         valid_idxs = [i for i, det in enumerate(detections) if det > 0.5]
180         landmarks = np.asarray(landmarks)[valid_idxs]
181         visibilities = np.asarray(visibilities)[valid_idxs]
182         poses = np.asarray(poses)[valid_idxs]
183         genders = np.asarray(genders)[valid_idxs]
184
185         # ========== Landmark-based NMS ==========
186         res_idx_sets = list()
187         precise_rects = [_bounding_region(l, v) for l, v
188                          in zip(landmarks, visibilities)]
189         areas = [common.rect_area(rect) for rect in precise_rects]
190         overlap_tls = 0.20  # TODO configure
191         scorebase_idxs = np.argsort(areas).tolist()  # ascending order
192         while len(scorebase_idxs) > 0:
193             # Register new index set with the best rect index
194             best_rect_idx = scorebase_idxs.pop()
195             res_idx_sets.append([best_rect_idx])
196             # Register overlapped indices
197             best_rect = precise_rects[best_rect_idx]
198
199 #             # [DEBUG] Draw L-NMS rectabgles
200 #             drawing._draw_rect(img, best_rect, (0, 1, 0))
201
202             removal_scorebase_idxs = list()
203             for s_i, i in enumerate(scorebase_idxs):
204                 overlap = common.rect_overlap_rate(best_rect, precise_rects[i])
205                 if overlap > overlap_tls:
206                     res_idx_sets[-1].append(i)
207                     removal_scorebase_idxs.append(s_i)
208             # Remove registered indices (reverse)
209             for i in removal_scorebase_idxs[::-1]:
210                 scorebase_idxs.pop(i)
211         # Extract middle value
212         res_landmarks = list()
213         res_visibilities = list()
214         res_poses = list()
```

```
215         res_genders = list()
216         res_rects = list()
217         for idx_set in res_idx_sets:
218             res_landmarks.append(np.median(landmarks[idx_set], axis=0))
219             res_visibilities.append(np.median(visibilities[idx_set], axis=0))
220             res_poses.append(np.median(poses[idx_set], axis=0))
221             res_genders.append(np.median(genders[idx_set], axis=0))
222             res_rects.append(_proposal_region(res_landmarks[-1],
223                                               res_visibilities[-1], img_rect))
224 
225         return (res_landmarks, res_visibilities, res_poses, res_genders,
226                 res_rects)
```

Arquivo common.py

```
1   # -*- coding: utf-8 -*-
2   import cv2
3   import dlib
4   import math
5   
6   # logging
7   from logging import getLogger, NullHandler
8   logger = getLogger(__name__)
9   logger.addHandler(NullHandler())
10  
11  
12  def _scale_down_image(img, max_img_size):
13      org_h, org_w = img.shape[0:2]
14      h, w = img.shape[0:2]
15      if max_img_size[0] < w:
16          h *= float(max_img_size[0]) / float(w)
17          w = max_img_size[0]
18      if max_img_size[1] < h:
19          w *= float(max_img_size[1]) / float(h)
20          h = max_img_size[1]
21      # Apply resizing
22      if h == org_h and w == org_w:
23          resize_scale = 1
24      else:
25          resize_scale = float(org_h) / float(h)  # equal to `org_w / w`
26          img = cv2.resize(img, (int(w), int(h)))
27      return img, resize_scale
28  
```

```
29
30  def selective_search_dlib(img, max_img_size=(500, 500),
31                            kvals=(50, 200, 2), min_size=2200, check=True,
32                            debug_window=False):
33      if debug_window:
34          org_img = img
35      org_h, org_w = img.shape[0:2]
36
37      # Resize the image for speed up
38      img, resize_scale = _scale_down_image(img, max_img_size)
39
40      # Selective search
41      drects = []
42      dlib.find_candidate_object_locations(img, drects, kvals=kvals,
43                                           min_size=min_size)
44      rects = [(int(drect.left() * resize_scale),
45                int(drect.top() * resize_scale),
46                int(drect.width() * resize_scale),
47                int(drect.height() * resize_scale)) for drect in drects]
48
49      # Check the validness of the rectangles
50      if check:
51          if len(rects) == 0:
52              logger.error('No selective search rectangle '
53                           '(Please tune the parameters)')
54          for rect in rects:
55              x, y = rect[0], rect[1]
56              w, h = rect[2], rect[3]
57              x2, y2 = x + w, y + h
58              if x < 0 or y < 0 or org_w < x2 or org_h < y2 or w <= 0 or h <= 0:
59                  logger.error('Invalid selective search rectangle, rect:{}, '
60                               'image:{}'.format(rect, (org_h, org_w)))
61
62      # Debug window
63      if debug_window:
64          for rect in rects:
65              p1 = (rect[0], rect[1])
66              p2 = (rect[0] + rect[2], rect[1] + rect[3])
67              cv2.rectangle(org_img, p1, p2, (0, 255, 0))
68          cv2.imshow('selective_search_dlib', org_img)
69          cv2.waitKey(0)
70
71      return rects
```

```
72
73
74  def rect_or(a, b):
75      x = min(a[0], b[0])
76      y = min(a[1], b[1])
77      w = max(a[0] + a[2], b[0] + b[2]) - x
78      h = max(a[1] + a[3], b[1] + b[3]) - y
79      return (x, y, w, h)
80
81
82  def rect_and(a, b):
83      x = max(a[0], b[0])
84      y = max(a[1], b[1])
85      w = min(a[0] + a[2], b[0] + b[2]) - x
86      h = min(a[1] + a[3], b[1] + b[3]) - y
87      if w < 0 or h < 0:
88          return (0, 0, 0, 0)
89      return (x, y, w, h)
90
91
92  def rect_area(a):
93      return a[2] * a[3]
94
95
96  def rect_overlap_rate(a, b):
97      area_and = rect_area(rect_and(a, b))
98      area_or = rect_area(rect_or(a, b))
99      if area_or == 0:
100         return 0
101     else:
102         return math.sqrt(float(area_and) / float(area_or))
```

Arquivo datasets.py

```
1   # -*- coding: utf-8 -*-
2   import cv2
3   import numpy as np
4   import os.path
5   import random
6   import sqlite3
7
8   import chainer
9
```

```
10  import common
11
12  # logging
13  from logging import getLogger, NullHandler
14  logger = getLogger(__name__)
15  logger.addHandler(NullHandler())
16
17  # Constant variables
18  N_LANDMARK = 21
19  IMG_SIZE = (227, 227)
20
21  # Python 2 compatibility
22  try:
23      FileNotFoundError
24  except NameError:
25      FileNotFoundError = IOError
26
27
28  def _exec_sqlite_query(cursor, select_str, from_str=None, where_str=None):
29      query_str = 'SELECT {}'.format(select_str)
30      query_str += ' FROM {}'.format(from_str)
31      if where_str:
32          query_str += ' WHERE {}'.format(where_str)
33      return [row for row in cursor.execute(query_str)]
34
35
36  def _load_raw_aflw(sqlite_path, image_dir):
37      ''' Load raw AFLW dataset from sqlite file
38      Return:
39          [dict('face_id', 'img_path', 'rect', 'landmark', 'landmark_visib',
40                'pose', 'gender')]
41      '''
42      logger.info('Load raw AFLW dataset from "{}"'.format(sqlite_path))
43
44      # Temporary dataset variables
45      dataset_dict = dict()
46
47      # Open sqlite file
48      conn = sqlite3.connect(sqlite_path)
49      cursor = conn.cursor()
50
51      # Basic property
52      select_str = "faces.face_id, imgs.filepath, " \
```

```
53                  "rect.x, rect.y, rect.w, rect.h, " \
54                  "pose.roll, pose.pitch, pose.yaw, metadata.sex"
55      from_str = "faces, faceimages imgs, facerect rect, facepose pose, " \
56                 "facemetadata metadata"
57      where_str = "faces.file_id = imgs.file_id and " \
58                  "faces.face_id = rect.face_id and " \
59                  "faces.face_id = pose.face_id and " \
60                  "faces.face_id = metadata.face_id"
61      query_res = _exec_sqlite_query(cursor, select_str, from_str, where_str)
62      # Register to dataset_dict
63      for face_id, path, rectx, recty, rectw, recth, roll, pitch, yaw, gender\
64              in query_res:
65          # Data creation or conversion
66          img_path = os.path.join(image_dir, path) if image_dir else path
67          landmark = np.zeros((N_LANDMARK, 2), dtype=np.float32)
68          landmark_visib = np.zeros(N_LANDMARK, dtype=np.float32)
69          pose = np.array([roll, pitch, yaw], dtype=np.float32)
70          gender = np.array(0 if gender == 'm' else 1, dtype=np.int32)
71          others_landmark_pts = list()
72          # Register
73          data = {'face_id': face_id,
74                  'img_path': img_path,
75                  'rect': (rectx, recty, rectw, recth),
76                  'landmark': landmark,
77                  'landmark_visib': landmark_visib,
78                  'pose': pose,
79                  'gender': gender,
80                  'others_landmark_pts': others_landmark_pts}
81          dataset_dict[face_id] = data
82
83      # Landmark property
84      # (Visibility is expressed by lack of the coordinate's row.)
85      select_str = "faces.face_id, coords.feature_id, " \
86                   "coords.x, coords.y"
87      from_str = "faces, featurecoords coords"
88      where_str = "faces.face_id = coords.face_id"
89      query_res = _exec_sqlite_query(cursor, select_str, from_str, where_str)
90      # Register to dataset_dict
91      invalid_face_ids = list()
92      for face_id, feature_id, x, y in query_res:
93          assert(1 <= feature_id <= N_LANDMARK)
94          if face_id in dataset_dict:
95              idx = feature_id - 1
```

```
96              dataset_dict[face_id]['landmark'][idx][0] = x
97              dataset_dict[face_id]['landmark'][idx][1] = y
98              dataset_dict[face_id]['landmark_visib'][idx] = 1
99          elif face_id not in invalid_face_ids:
100             logger.warn('Invalid face id ({}) in AFLW'.format(face_id))
101             invalid_face_ids.append(face_id)
102
103     # Landmarks of other faces
104     select_str = "a.face_id, coords.x, coords.y"
105     from_str = "faces a, faces b, featurecoords coords"
106     where_str = "a.face_id != b.face_id and a.file_id = b.file_id and " \
107                 "b.face_id = coords.face_id"
108     query_res = _exec_sqlite_query(cursor, select_str, from_str, where_str)
109     # Register to dataset_dict
110     for face_id, others_x, others_y in query_res:
111         if face_id in dataset_dict:
112             other_coord = [others_x, others_y]
113             dataset_dict[face_id]['others_landmark_pts'].append(other_coord)
114         else:
115             assert(face_id in invalid_face_ids)
116     # Convert list to np.ndarray
117     for data in dataset_dict.values():
118         pts = np.array(data['others_landmark_pts'], dtype=np.float32)
119         data['others_landmark_pts'] = pts
120
121     # Exit sqlite
122     cursor.close()
123
124     # Return dataset_dict's value (list)
125     return list(dataset_dict.values())
126
127
128 def _rect_contain(rect, pt):
129     x, y, w, h = rect
130     return x <= pt[0] <= x + w and y <= pt[1] <= y + h
131
132
133 def _extract_valid_rects(rects, img, others_landmark_pts):
134     ''' Extract rectangles which do not contain other landmarks '''
135     # Extraction
136     dst = list()
137     for rect in rects:
138         # Check if others landmarks are contained
```

```
139             for others_pt in others_landmark_pts:
140                 if _rect_contain(rect, others_pt):
141                     break
142             else:
143                 dst.append(rect)
144 
145         # avoid no rectangle
146         if len(dst) == 0:
147             dst.append((0, 0, img.shape[1], img.shape[0]))
148 
149         return dst
150 
151 
152     def _flip_y(img, landmark, landmark_visib, pose):
153         # copy
154         img = np.array(img)
155         landmark = np.array(landmark)
156         pose = np.array(pose)
157 
158         # Flip
159         img = img[:, ::-1, :]
160 
161         # AFLW 21 points landmark
162         #  0|LeftBrowLeftCorner
163         #  1|LeftBrowCenter
164         #  2|LeftBrowRightCorner
165         #  3|RightBrowLeftCorner
166         #  4|RightBrowCenter
167         #  5|RightBrowRightCorner
168         #  6|LeftEyeLeftCorner
169         #  7|LeftEyeCenter
170         #  8|LeftEyeRightCorner
171         #  9|RightEyeLeftCorner
172         #  10|RightEyeCenter
173         #  11|RightEyeRightCorner
174         #  12|LeftEar
175         #  13|NoseLeft
176         #  14|NoseCenter
177         #  15|NoseRight
178         #  16|RightEar
179         #  17|MouthLeftCorner
180         #  18|MouthCenter
181         #  19|MouthRightCorner
```

```
182         #  20|ChinCenter
183         corres_dict = {
184             0: 5, 1: 4, 2: 3, 3: 2, 4: 1, 5: 0,  # brow
185             6: 11, 7: 10, 8: 9, 9: 8, 10: 7, 11: 6,  # eye
186             12: 16, 16: 12, 20: 20,  # ear and chin
187             13: 15, 14: 14, 15: 13,  # nose
188             17: 19, 18: 18, 19: 17  # mouse
189         }
190         dst_landmark = np.zeros_like(landmark)
191         dst_landmark_visib = np.zeros_like(landmark_visib)
192         for i, (pt, visib) in enumerate(zip(landmark, landmark_visib)):
193             dst_i = corres_dict[i]
194             dst_landmark[dst_i][0] = pt[0] * -1.0
195             dst_landmark[dst_i][1] = pt[1]
196             dst_landmark_visib[dst_i] = visib
197         landmark = dst_landmark
198         landmark_visib = dst_landmark_visib
199 
200         # pose (x, y, z), (roll, pitch yaw)
201         pose[0] *= -1.0  # x
202         pose[2] *= -1.0  # z
203 
204         return img, landmark, landmark_visib, pose
205 
206 
207     class AFLW(chainer.dataset.DatasetMixin):
208         ''' AFLW Dataset
209         Arguments
210             * n_try_detect_alt(int):
211                 The number of trying an alternation of the detection when
212                 get_example() is called. To disable the alternation, set 1.
213             * overlap_tls(dict):
214                 Overlap thresholds of face rectangles and selective search's ones.
215             * random_flip(bool):
216                 A flag for random flip of datasets.
217             * min_valid_landmark_cnt(int):
218                 The least number for valid landmark. This value is used for
219                 deciding whether faces and their landmarks are valid.
220         '''
221 
222         def __init__(self, n_try_detect_alt=30,
223                      overlap_tls={'detection_p': 0.50, 'detection_n': 0.35,
224                                   'landmark': 0.35, 'pose': 0.50, 'gender': 0.50},
```

```
225                  random_flip=True, min_valid_landmark_cnt=3):
226         chainer.dataset.DatasetMixin.__init__(self)
227
228         # Member variables
229         self.dataset = list()
230         self.detect_alt_diff = 0
231         self.n_try_detect_alt = n_try_detect_alt
232         self.overlap_tls = overlap_tls
233         self.random_flip = random_flip
234         self.min_valid_landmark_cnt = min_valid_landmark_cnt
235         self.raw_mode = False
236
237     def setup_raw(self, sqlite_path, image_dir, log_interval=10):
238         # Load raw AFLW dataset
239         self.dataset = _load_raw_aflw(sqlite_path, image_dir)
240
241         # Calculate selective search rectangles (This takes many minutes)
242         logger.info('Calculate selective search rectangles for AFLW')
243         for i, entry in enumerate(self.dataset):
244             # Logging
245             if i % log_interval == 0:
246                 logger.info(' {}/{}'.format(i, len(self.dataset)))
247
248             # Load image
249             img = cv2.imread(entry['img_path'])
250             if img is None or img.size == 0:
251                 # Empty elements
252                 logger.warn('Failed to load image {}'.format(
253                     entry['img_path']))
254                 self.dataset[i]['ssrects'] = list()
255                 self.dataset[i]['ssrect_overlaps'] = list()
256             else:
257                 # Selective search
258                 ssrects = common.selective_search_dlib(img)
259                 ssrects = _extract_valid_rects(ssrects, img,
260                                                entry['others_landmark_pts'])
261                 overlaps = [common.rect_overlap_rate(ssrect, entry['rect'])
262                             for ssrect in ssrects]
263                 self.dataset[i]['ssrects'] = ssrects
264                 self.dataset[i]['ssrect_overlaps'] = overlaps
265
266     def __len__(self):
267         return len(self.dataset)
```

```
268
269         def get_example(self, i):
270             entry = self.dataset[i]
271
272             # Raw entry
273             if self.raw_mode:
274                 return entry
275
276             # Loop for detection alternation
277             try_cnt = 0
278             special_skip_cnt = 0
279             while try_cnt < self.n_try_detect_alt:
280                 try_cnt += 1
281
282                 # === Entry variables ===
283                 img_path = entry['img_path']
284                 landmark = entry['landmark']
285                 landmark_visib = entry['landmark_visib']
286                 pose = entry['pose']
287                 gender = entry['gender']
288                 ssrects = entry['ssrects']
289                 overlaps = entry['ssrect_overlaps']
290                 # Random ssrect
291                 if len(ssrects) == 0:
292                     logger.warn('No selective search rectangle')
293                     raise IndexError
294                 ssrect_idx = random.randint(0, len(ssrects) - 1)
295                 ssrect = ssrects[ssrect_idx]
296                 overlap = overlaps[ssrect_idx]
297                 x, y, w, h = ssrect
298
299                 # === Crop and Normalize 1 (landmark) ===
300                 # Landmark ([-0.5:0.5])
301                 landmark_offset = np.array(
302                     [x + w / 2, y + h / 2], dtype=np.float32)
303                 landmark_denom = np.array([w, h], dtype=np.float32)
304                 landmark = (landmark - landmark_offset) / landmark_denom
305                 # Consider range of the cropped rectangle
306                 hidden_idxs = np.where((landmark < -0.5) | (0.5 < landmark))
307                 landmark = landmark.copy()
308                 landmark[hidden_idxs[0]] = 0.0  # mask [x, y] (overwrite)
309                 landmark_visib = landmark_visib.copy()
310                 landmark_visib[hidden_idxs[0]] = 0.0  # mask (overwrite)
```

```
311
312             # === Convert 1 (Detection) ===
313             if overlap > self.overlap_tls['detection_p']:
314                 detection = np.array(1, dtype=np.int32)
315             elif overlap < self.overlap_tls['detection_n']:
316                 detection = np.array(0, dtype=np.int32)
317             else:
318                 detection = np.array(-1, dtype=np.int32)  # Ignore
319
320             # === Special Skip for invalid landmark faces ===
321             n_valid_landmark = landmark_visib[landmark_visib > 0.5].shape[0]
322             if detection == 1 and \
323                n_valid_landmark < self.min_valid_landmark_cnt:
324                 try_cnt -= 1
325                 special_skip_cnt += 1
326                 if special_skip_cnt > 10:  # avoid infinity loop
327                     break
328                 else:
329                     continue
330
331             # === Check the alternation ===
332             if detection == -1:
333                 continue
334             if self.detect_alt_diff > 0 and detection == 0:  # Negative sample
335                 self.detect_alt_diff -= 1
336                 break
337             if self.detect_alt_diff <= 0 and detection == 1:  # Positive sample
338                 self.detect_alt_diff += 1
339                 break
340         # End of detection alternation loop
341
342         # === Crop and Normalize 2 (image) ===
343         img = cv2.imread(img_path)
344         if img is None or img.size == 0:
345             logger.warn('Invalid image "{}"'.format(img_path))
346             raise IndexError
347         # Image
348         img = img[y:y + h + 1, x:x + w + 1, :]
349         if img.size == 0:
350             org_img = cv2.imread(img_path)
351             logger.warn('Invalid crop rectangle. (rect:{}, img:{}, org_img{}")'
352                         .format(ssrect, img.shape, org_img.shape))
353             raise IndexError
```

```
354             img = cv2.resize(img, IMG_SIZE)
355             img = img.astype(np.float32)
356             # [0:255](about) -> [-0.5:0.5]
357             img = cv2.normalize(img, None, -0.5, 0.5, cv2.NORM_MINMAX)
358
359             # === Random flip ===
360             if self.random_flip and random.randint(0, 1):
361                 flip_ret = _flip_y(img, landmark, landmark_visib, pose)
362                 img, landmark, landmark_visib, pose = flip_ret
363
364             # === Convert 2 (and mask) ===
365             # Image
366             img = np.transpose(img, (2, 0, 1))
367             # Landmark and visibility
368             if overlap > self.overlap_tls['landmark']:
369                 # [21, 2] -> [42]
370                 landmark = landmark.reshape(42)
371                 # Mask
372                 mask_landmark = np.ones_like(landmark)
373                 mask_landmark_visib = np.ones_like(landmark_visib)
374             else:
375                 # [21, 2] -> [42]
376                 landmark = landmark.reshape(42)
377                 # No difference
378                 landmark = np.zeros_like(landmark)
379                 landmark_visib = np.zeros_like(landmark_visib)
380                 mask_landmark = np.zeros_like(landmark)
381                 mask_landmark_visib = np.zeros_like(landmark_visib)
382             # Pose
383             if overlap > self.overlap_tls['pose']:
384                 mask_pose = np.ones_like(pose)
385             else:
386                 # No difference
387                 pose = np.zeros_like(pose)
388                 mask_pose = np.zeros_like(pose)
389             # Gender
390             if overlap > self.overlap_tls['gender']:
391                 pass  # use entry value
392             else:
393                 gender = np.array(-1, dtype=np.int32)  # Ignore
394
395             return {'x_img': img, 't_detection': detection, 't_landmark': landmark,
396                     't_visibility': landmark_visib, 't_pose': pose,
```

```
                'm_visibility': mask_landmark_visib, 'm_pose': mask_pose}


def setup_aflw(cache_path, sqlite_path=None, image_dir=None, test_rate=0.04,
               raw_mode=False):
    # Empty AFLW
    aflw = AFLW()
    aflw.raw_mode = raw_mode

    logger.info('Try to load AFLW cache from "{}"'.format(cache_path))
    try:
        # Load cache
        cache_data = np.load(cache_path)
        dataset = cache_data['dataset'].tolist()
        n_train = int(cache_data['n_train'])
        order = cache_data['order'].tolist()
        # Set to AFLW
        aflw.dataset = dataset
        logger.info('Succeeded in loading AFLW cache')

    except (FileNotFoundError, KeyError):
        # Setup AFLW
        logger.info('Failed to load AFLW cache, so setup now')
        if not sqlite_path:
            logger.critical('`sqlite_path` is needed to load raw AFLW')
        aflw.setup_raw(sqlite_path, image_dir)

        # Generate order to split into train/test
        n_train = int(len(aflw.dataset) * (1.0 - test_rate))
        order = np.random.permutation(len(aflw.dataset))

        # Save cache
        logger.info('Save AFLW cache to "{}"'.format(cache_path))
        np.savez(cache_path, dataset=aflw.dataset,
                 n_train=n_train, order=order)

    # Split dataset into train and test
    train, test = chainer.datasets.split_dataset(aflw, n_train, order=order)
    logger.info('AFLW datasets (n_train:{}, n_test:{})'.
                format(len(train), len(test)))

    return train, test
```

```
440
441
442 def aflw_template_landmark():
443     ''' Template landmark from AFLW dataset '''
444     return np.array([[-0.479962468147, 0.471864163876],
445                      [-0.30303606391, 0.508996844292],
446                      [-0.106451146305, 0.498075485229],
447                      [0.106451146305, 0.498075485229],
448                      [0.30303606391, 0.508996844292],
449                      [0.479962468147, 0.471864163876],
450                      [-0.447198301554, 0.321149080992],
451                      [-0.318325966597, 0.325517624617],
452                      [-0.163242310286, 0.308043420315],
453                      [0.163242310286, 0.308043420315],
454                      [0.318325966597, 0.325517624617],
455                      [0.447198301554, 0.321149080992],
456                      [-0.674257874489, -0.151652157307],
457                      [-0.170000001788, -0.075740583241],
458                      [0.0, 0.0],
459                      [0.170000001788, -0.075740583241],
460                      [0.674257874489, -0.151652157307],
461                      [-0.272456139326, -0.347239643335],
462                      [0.0, -0.336318254471],
463                      [0.272456139326, -0.347239643335],
464                      [0.0, -0.737950384617]], dtype=np.float32)
```

Arquivo drawing.py

```
1  # -*- coding: utf-8 -*-
2  import cv2
3  import numpy as np
4  import math
5
6  # logging
7  from logging import getLogger, NullHandler
8  logger = getLogger(__name__)
9  logger.addHandler(NullHandler())
10
11 # matplotlib
12 try:
13     import matplotlib
14     matplotlib.use('Agg')
15     import matplotlib.pyplot as plt
```

```
    import matplotlib.backends.backend_agg as agg
except Exception as e:
    print('error')
    logger.error('Failed to import matplotlib')
    logger.error('[%s] %s', str(type(e)), str(e.args))
    exit()


def _draw_line(img, pt1, pt2, color, thickness=2):
    pt1 = (int(pt1[0]), int(pt1[1]))
    pt2 = (int(pt2[0]), int(pt2[1]))
    cv2.line(img, pt1, pt2, color, int(thickness))


def _draw_circle(img, pt, color, radius=4, thickness=-1):
    pt = (int(pt[0]), int(pt[1]))
    cv2.circle(img, pt, radius, color, int(thickness))


def _draw_rect(img, rect, color, thickness=2):
    p1 = (int(rect[0]), int(rect[1]))
    p2 = (int(rect[0] + rect[2]), int(rect[1] + rect[3]))
    cv2.rectangle(img, p1, p2, color, thickness)


def _draw_cross(img, pt, color, size=4, thickness=2):
    p0 = (pt[0] - size, pt[1] - size)
    p1 = (pt[0] + size, pt[1] + size)
    p2 = (pt[0] + size, pt[1] - size)
    p3 = (pt[0] - size, pt[1] + size)
    _draw_line(img, p0, p1, color, thickness)
    _draw_line(img, p2, p3, color, thickness)


def _rotation_matrix(rad_x, rad_y, rad_z):
    cosx, cosy, cosz = math.cos(rad_x), math.cos(rad_y), math.cos(rad_z)
    sinx, siny, sinz = math.sin(rad_x), math.sin(rad_y), math.sin(rad_z)
    rotz = np.array([[cosz, -sinz, 0],
                     [sinz, cosz, 0],
                     [0, 0, 1]], dtype=np.float32)
    roty = np.array([[cosy, 0, siny],
                     [0, 1, 0],
                     [-siny, 0, cosy]], dtype=np.float32)
```

```
59      rotx = np.array([[1, 0, 0],
60                       [0, cosx, -sinx],
61                       [0, sinx, cosx]], dtype=np.float32)
62      return rotx.dot(roty).dot(rotz)
63
64
65  def _project_plane_yz(vec):
66      x = vec.dot(np.array([0, 1, 0], dtype=np.float32))
67      y = vec.dot(np.array([0, 0, 1], dtype=np.float32))
68      return np.array([x, -y], dtype=np.float32)  # y flip
69
70
71  def draw_detection(img, detection, size=15):
72      # Upper left
73      pt = (size + 5, size + 5)
74      if detection:
75          _draw_circle(img, pt, (0, 200, 0), size, 5)
76      else:
77          _draw_cross(img, pt, (0, 0, 200), size, 5)
78
79
80  def draw_landmark(img, landmark, visibility, color, line_color_scale,
81                    denormalize_scale=True):
82      """  Draw AFLW 21 points landmark
83          0|LeftBrowLeftCorner
84          1|LeftBrowCenter
85          2|LeftBrowRightCorner
86          3|RightBrowLeftCorner
87          4|RightBrowCenter
88          5|RightBrowRightCorner
89          6|LeftEyeLeftCorner
90          7|LeftEyeCenter
91          8|LeftEyeRightCorner
92          9|RightEyeLeftCorner
93          10|RightEyeCenter
94          11|RightEyeRightCorner
95          12|LeftEar
96          13|NoseLeft
97          14|NoseCenter
98          15|NoseRight
99          16|RightEar
100         17|MouthLeftCorner
101         18|MouthCenter
```

```
        19|MouthRightCorner
        20|ChinCenter
    """
    conn_list = [[0, 1], [1, 2], [3, 4], [4, 5],  # brow
                 [6, 7], [7, 8], [9, 10], [10, 11],  # eye
                 [13, 14], [14, 15], [13, 15],  # nose
                 [17, 18], [18, 19],  # mouse
                 [12, 20], [16, 20]]  # face contour

    if landmark.ndim == 1:
        landmark = landmark.reshape(int(landmark.shape[-1] / 2), 2)
    assert(landmark.shape[0] == 21 and visibility.shape[0] == 21)

    if denormalize_scale:
        h, w = img.shape[0:2]
        size = np.array([[w, h]], dtype=np.float32)
        landmark = landmark * size + size / 2

    # Line
    line_color = tuple(v * line_color_scale for v in color)
    for i0, i1 in conn_list:
        if visibility[i0] > 0.5 and visibility[i1] > 0.5:
            _draw_line(img, landmark[i0], landmark[i1], line_color, 2)

    # Point
    for pt, visib in zip(landmark, visibility):
        if visib > 0.5:
            _draw_circle(img, pt, color, 4, -1)
        else:
            _draw_circle(img, pt, color, 4, 1)


def draw_pose(img, pose, size=30, idx=0):
    # parallel projection (something wrong?)
    rotmat = _rotation_matrix(-pose[0], -pose[1], -pose[2])
    zvec = np.array([0, 0, 1], np.float32)
    yvec = np.array([0, 1, 0], np.float32)
    xvec = np.array([1, 0, 0], np.float32)
    zvec = _project_plane_yz(rotmat.dot(zvec))
    yvec = _project_plane_yz(rotmat.dot(yvec))
    xvec = _project_plane_yz(rotmat.dot(xvec))

    # Lower left
```

```
145         org_pt = ((size + 5) * (2 * idx + 1), img.shape[0] - size - 5)
146         _draw_line(img, org_pt, org_pt + zvec * size, (255, 0, 0), 3)
147         _draw_line(img, org_pt, org_pt + yvec * size, (0, 255, 0), 3)
148         _draw_line(img, org_pt, org_pt + xvec * size, (0, 0, 255), 3)
149
150     def draw_gender(img, gender, size=7, idx=0):
151         # Upper right
152         pt = (img.shape[1] - (size + 5) * (2 * idx + 1), size + 5)
153         if gender == 0:
154             _draw_circle(img, pt, (255, 0, 0), size, -1)  # male
155         elif gender == 1:
156             _draw_circle(img, pt, (0, 0, 255), size, -1)  # female
157
158
159     def draw_gender_rect(img, gender, rect):
160         if gender == 0:
161             _draw_rect(img, rect, (1.0, 0.3, 0.3))  # male
162         elif gender == 1:
163             _draw_rect(img, rect, (0.3, 0.3, 1.0))  # female
164
165
166     def draw_loss_graph(train_loss_list, test_loss_list, train_epoch_list=None,
167                         test_epoch_list=None, train_color='blue', test_color='red',
168                         legend_loc='upper right', title=None):
169         # Axis data
170         # Losses
171         train_loss = np.asarray(train_loss_list)
172         test_loss = np.asarray(test_loss_list)
173         # Epochs
174         if train_epoch_list:
175             train_epoch = np.asarray(train_epoch_list)
176         else:
177             train_epoch = np.arange(0, len(train_loss_list))
178         if test_epoch_list:
179             test_epoch = np.asarray(test_epoch_list)
180         else:
181             test_epoch = np.arange(0, len(test_loss_list))
182
183         # Create new figure
184         plt.clf()
185         fig, ax = plt.subplots()
186         ax.plot(train_epoch, train_loss, label='train', color=train_color)
187         ax.plot(test_epoch, test_loss, label='test', color=test_color)
```

```
188
189     def draw_annotate(label, x, y, color):
190         ax.scatter(x, y, 20, color=color)
191         ax.annotate(label, xy=(x, y), xytext=(+20, +10),
192                     textcoords='offset points',
193                     arrowprops={'arrowstyle': '->',
194                                 'connectionstyle': 'arc3,rad=.2'})
195
196     # Show min values
197     if train_loss.shape[0] > 0:
198         min_idx = np.argmin(train_loss)
199         x, y = train_epoch[min_idx], train_loss[min_idx]
200         draw_annotate('min train loss: %0.3f' % y, x, y, train_color)
201     if test_loss.shape[0] > 0:
202         min_idx = np.argmin(test_loss)
203         x, y = test_epoch[min_idx], test_loss[min_idx]
204         draw_annotate('min test loss: %0.3f' % y, x, y, test_color)
205
206     # Settings
207     ax.set_xlabel("epoch")
208     ax.set_ylabel("loss rate")
209     ax.set_xlim(left=0)
210     ax.set_ylim(bottom=0)
211     ax.legend(loc=legend_loc)
212     if title is not None:
213         ax.set_title(title)
214
215     # Draw
216     canvas = agg.FigureCanvasAgg(fig)
217     canvas.draw()
218     renderer = canvas.get_renderer()
219     img = np.fromstring(renderer.tostring_rgb(), dtype=np.uint8, sep='')
220     img = img.reshape(canvas.get_width_height()[::-1] + (3,))
221
222     # Close
223     plt.close('all')
224
225     return img
```

Arquivo config.py

```
1  # -*- coding: utf-8 -*-
2  import json
```

```
3
4   # logging
5   from logging import getLogger, NullHandler
6   logger = getLogger(__name__)
7   logger.addHandler(NullHandler())
8
9
10  def load(filename):
11      ''' Load configure json file
12      Loaded variables will be stored in config's globals
13      '''
14      logger.info('Load config from "{}"'.format(filename))
15      f = open(filename)
16      data = json.load(f)
17      # parse json to python variables
18      for key, value in data.items():
19          if key in globals():
20              logger.error('Conflict in config with key "{}"'.format(key))
21          else:
22              globals()[key] = value
```

APÊNDICE B – Artigo

RECONHECIMENTO DE MOVIMENTAÇÃO CORPORAL UTILIZANDO REDES NEUARIS

Thiago D. da Silveira

Universidade Federal de Santa Catarina (UFSC)
Departamento de Informática e Estatística
Campus Universitário – Florianópolis – SC – Brazil

thiagods.ti@gmail.com

Abstract. *Artificial neural networks are computational models that imitates the central nervous system of a human. Today many companies are using artificial intelligence in areas such as: autonomous cars, security, games and even tools that help people with motor disabilities.*

This work aims to present a model of recognition of body movement using neural networks, make analyzes and test it.

Hyperface was the model used to make analyzes and to test it in differents videos: recorded in webcam, got from youtube and received from Salford University.

The result was satisfactory in general, but it was noticed that much of the analysis has interference by the dataset and the training used to learn the model.

Resumo. *Redes neurais artificiais são modelos computacionais que imitam o sistema nervoso central de um humano. Hoje muitas empresas estão utilizando inteligência artificial em áreas distintas como: carros autônomos, segurança, jogos e em até ferramentas que auxiliam pessoas com deficiências motoras.*

Este trabalho visa apresentar um modelo de reconhecimento de movimentação corporal utilizando redes neurais, fazer análises e teste do mesmo.

Foi utilizado o modelo Hyperface para fazer as análises e testes em diferentes vídeos: gravado em webcam, obtido do youtube e recebido da Universidade de Salford.

O resultado obtido foi satisfatório de forma geral, porém foi percebido que muito da análise tem interferência pelo dataset e o treinamento utilizado para o aprendizado do modelo.

1. Introdução

Dentro da inteligência artificial existem dois paradigmas mais representativos que são o simbólico, que segundo Russel e Norvig(2004, p. 20) foi apresentado por Newell e Simon em 1976. Esta representação utiliza os símbolos e outros conhecimentos explícitos, eles afirmam que todo sistema que tenha inteligência deve operar com uma estrutura de dados composta por símbolos - e conexionista - que representam o raciocínio com conexões como as redes neurais, que apresentam os neurônios e suas conexões como forma de racionalizar. Apesar das suas diferentes abordagens Russel e Norvig afirmam que elas devem ser vistas como complementares e não como concorrentes (2004, pg. 26).

Haykin (2001, p. 1) apresenta Redes Neurais como uma tecnologia que tem raízes em muitas disciplinas: neurociência, matemática, estatística, física, ciência da computação e engenharia. Possui uma propriedade muito importante que é a de aprender. Uma rede neuronal ou neural tem o intuito de representar como o cérebro humano realiza uma tarefa particular ou função de interesse.

Redes Neurais Artificiais são desenvolvidas com componentes eletrônicos ou então simuladas com programação em um computador. Uma Rede neural artificial (RNA) é composta de neurônios que são interligados entre eles, as conexões entre esses neurônios são chamadas de sinapses.

De acordo com Coppin (2013, p. 233) aprendizagem é um segmento extremamente importante da IA. Coppin (2013, p. 248) afirma que as abordagens mais relevantes de aprendizagem são por reforço, supervisionada e a não supervisionada.

A finalidade deste trabalho de conclusão de curso é reconhecer a movimentação corporal de uma pessoa que apareça no vídeo, o reconhecimento da movimentação será focado na direção do rosto da pessoa. Faremos testes com uma rede neural convolucional ou também conhecida como convolutiva, esse tipo de rede neural utiliza a aprendizagem supervisionada. Esse modelo segundo Haykin (2001, p. 271) serve para reconhecer formas bidimensionais com um alto grau de invariância quanto formas de distorções e por isso é utilizado para reconhecimento de imagens.

2. Conceitos Básicos

Esta seção tem como objetivo criar um embasamento teórico no que diz respeito as redes neurais e aprendizagem, sendo parte fundamental para o entendimento deste trabalho. Na seção 2.1 é apresentado a definação de uma rede neural, na seção 2.2 o aprendizado supervisionado, na seção 2.3 falaremos sobre rede neural convolucional e por fim na seção 2.4 explicaremos como é o reconhecimento de uma movimentação corporal da região da cabeça.

2.1 Rede Neural

O neurônio, segundo Coppin (2013), é um elemento de processamento que é composto de uma soma, que é o corpo do neurônio, um axônio e vários dendritos.

A comunicação entre neurônios ocorre da seguinte maneira. Primeiro um neurônio recebe um sinal de entrada através dos dendritos vindo de outros neurônios, quando esse sinal excede um limiar, então esse neurônio é ativado e envia seu sinal para outro neurônio através das ramificações terminais do axônio. Na computação é a mesma coisa. (COPPIN, 2013, p. 262).

Para reconhecer a movimentação corporal, que é um desafio complexo computacionalmente porém simples para um cérebro humano, primeiramente precisamos reconhecer um humano. Nguyen, Li e Ogunboma (2016) apud Wang (2017) citam que geralmente o processo de detecção de humano é através dos seguintes passos: extrair regiões de interesse de uma imagem, descrever as regiões com descritores e então classificá-las como humana ou não-humana.

2.2 Aprendizagem Supervisionada

Haykin (2001) afirma que a propriedade primordial para uma RNA é a aprendizagem. Está ocorre pela alteração dos pesos entre as conexões dos neurônios.

Existem os aprendizados supervisionados e não supervisionados, contudo iremos nos aprofundar a qual utilizamos nesse trabalho que é a aprendizagem supervisionada.

Quando uma rede neural é configurada para ter um aprendizado supervisionado, Haykin (2001) explica que em virtude de um conhecimento prévio, o professor pode dar à rede neural o resultado esperado, ou seja, o treinamento supervisionado precisa de um conjunto de dados já previamente classificado.

A aprendizagem supervisionada é simples, dado uma entrada, a rede neural da uma resposta. A diferença entre a resposta e o valor real (conhecido préviamente) é o ajuste que a rede neural terá que fazer. Esse processo é repetido algumas vezes, até que seja satisfeito o ponto de parada, que pode ser por quantidade de épocas ou pode ser um limite de acerto mínimo.

2.3 Rede Neural Convolucional

Redes neurais convolucionais (RNC) são um tipo específico de rede neural artificial, segundo Bengio, Goodfellow e Courville (2015) apud Wang (2017) no artigo *Deep Learning* em 2016, trata-se de usar convolução em pelo menos uma camada ao invés de operações com vetores e matrizes. Inicialmente a RNC foi criada para problemas de classificação de imagem.

Em 2012 foi criado uma RNC conhecida como AlexNet, esta obteve um grande sucesso no concurso *ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) 2012*, após isso os métodos para redes neurais convolucionais foram bem difundidas.

É importante apresentar a AlexNet, pois a RNC que foi utilizada no trabalho é baseado nela. Na Figura 1 podemos ver a arquiteura da AlexNet.

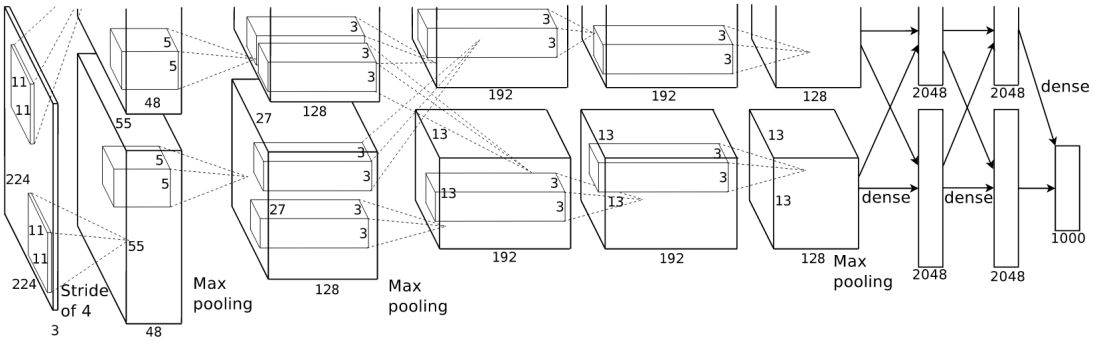


Figura 1. AlexNet arquitetura

2.4 Posição da Cabeça

Foi utilizado a anotação *pitch, yaw e roll* comumente usada para descrever os três graus de liberdade da posição da cabeça.

Conforme a figura 2 *pitch* e *yaw* indicam o deslocamento da posição da cabeça entre horizontal e vertical e *roll* é a angulação.

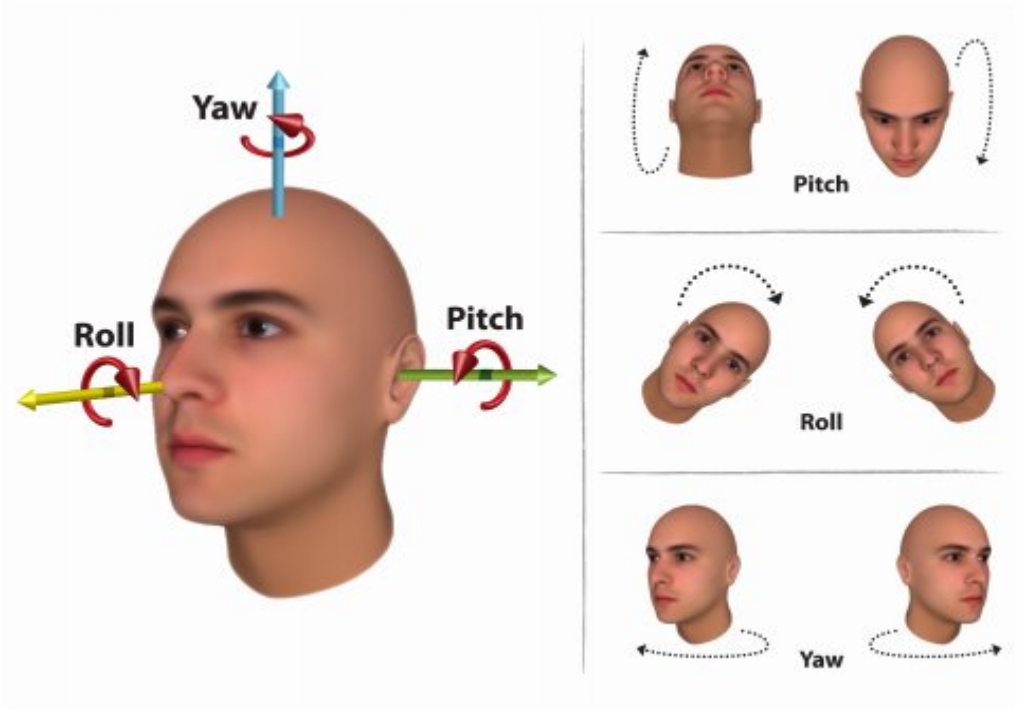


Figura 2. Pitch, Yaw e Roll

3. Hyperface

Esta seção explica o modelo que utilizamos para fazer o reconhecimento de movimentação corporal.

Hyperface segundo Ranjan, Patel e Chellappa (2016) é um algoritmo que usa rede neural convolucional para prover simultâneas funcionalidades como: detectar faces, localização de pontos relevantes da face, estimação da pose da face e reconhecimento de gênero em uma foto.

Inicialmente a *Hyperface* é composta pela rede neural AlexNet para classificação de imagem. Esta foi modificada para conter apenas 5 camadas convolucionais como podemos ver na Figura 3. A rede é treinada para a tarefa de detecção de faces.

Após o processamento é feita uma fusão das camadas da AlexNet para no final termos 5 camadas de saída representando: Detecção de face (face ou não face), Marcações faciais (pontos), Visibilidade (fator), Pose (*roll, pitch e yaw*) e Gênero (masculino ou feminino).

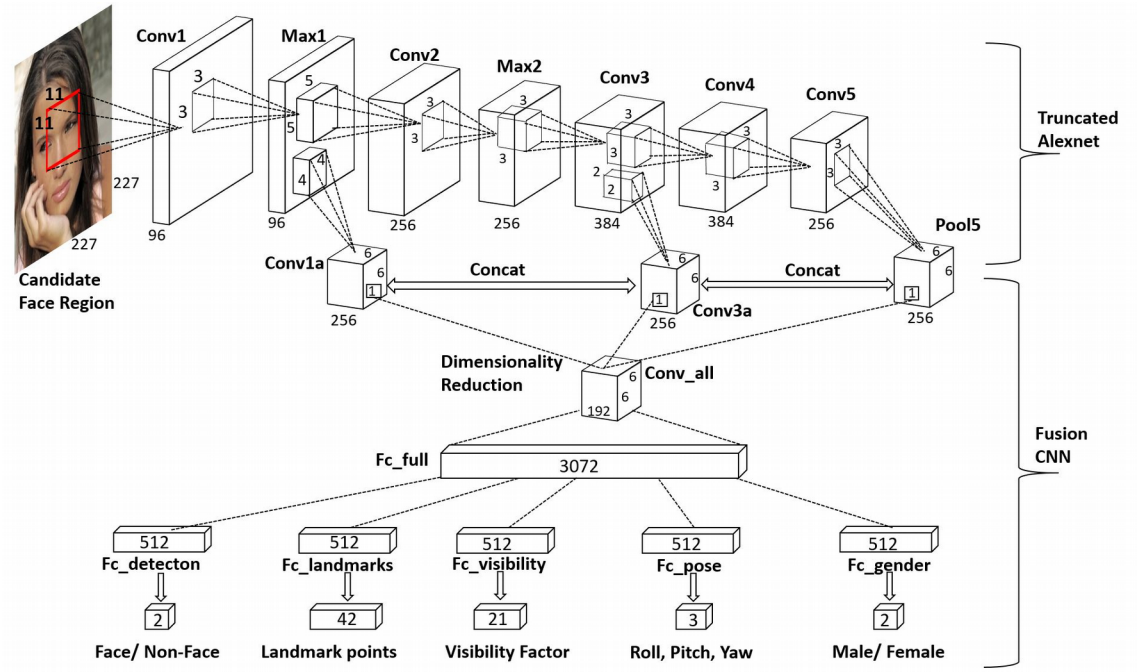


Figura 3. Hyperface arquitetura

Para o treinamento da rede foi utilizado o AFLW *Dataset,* este contém 25,993 imagens com marcações faciais, onde cada imagem tem 21 marcações. As imagens possuem variações de pose, expressão, etnia, idade e gênero. Foram utilizadas 1000 imagens para teste, e as outras para o treinamento da rede.

Para a estimativa da posição da pose da face foi utilizada a perda Euclidiana para estimar a roll (p1), pitch (p2) e yaw (p3), foi computado a perda por cada região da imagem obtendo uma sobreposição maior que 0.5.

$$loss = \frac{(p'1 - p1)^2 + (p'2 - p2)^2 + (p'3 - p3)^2}{3}$$

Como podemos ver na função descrita, o valor resultante de *loss* é dado pela média dos ângulos que são dados pela diferença elevado ao quadrado entre o valor de p' estimado e do valor de p real.

4. Análises e Experimentos

A análise dos resultados foram feitas por cada frame de um vídeo. Ns seção 4.1 mostraremos o resultado da análise feita para os videos da webcam. Seção 4.2 será mostrado a análise para vídeos do youtube, e por fim, na seção 4.3 a análise do vídeo de salford.

Na Figura 4 podemos ver que a triangulação do *pitch, yaw e roll* encontrasse no canto inferior esquerdo. A bolinha azul no canto superior direito indica o gênero da pessoa, podendo ser azul para masculino e vermelho para feminino. Também temos os pontos faciais, que quando estão em verde indicam que foram encontradas todas as anotações faciais.

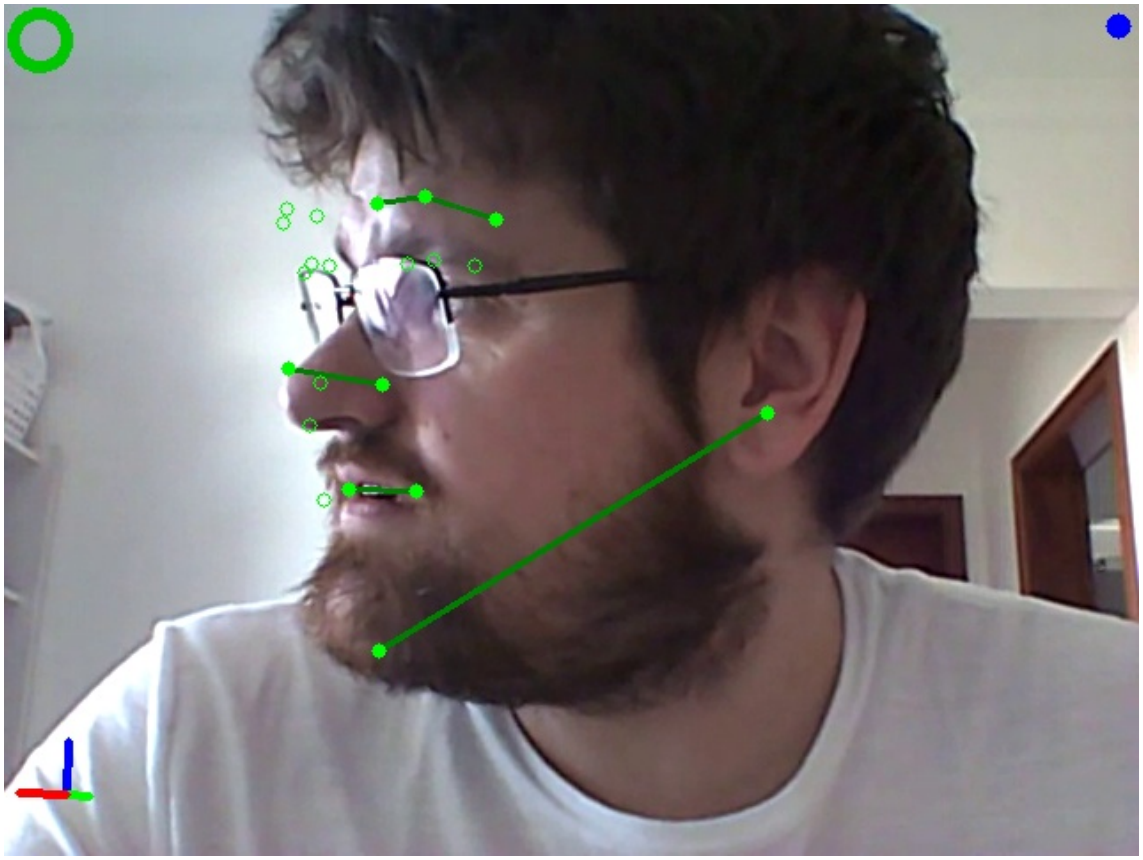


Figura 4. Hyperface análise

4.1 Análise Webcam

As melhores detecções que obtivemos foram de imagens gravadas da webcam. Como essas imagens estão mais parecidas com as imagens do *dataset* utilizado no treinamento o resultado consequentemente iria ser melhor, porém isso foi concluído ao final de todos as outras análises.

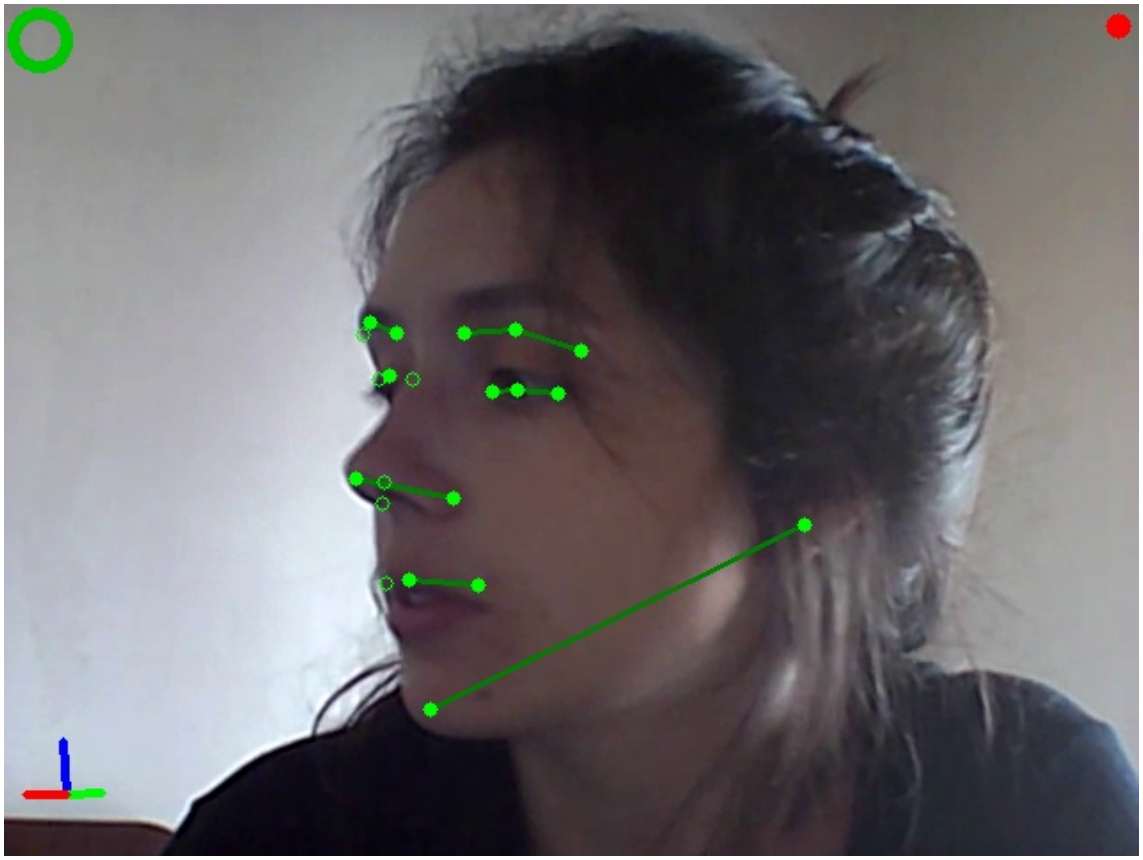


Figura 5. Webcam 1

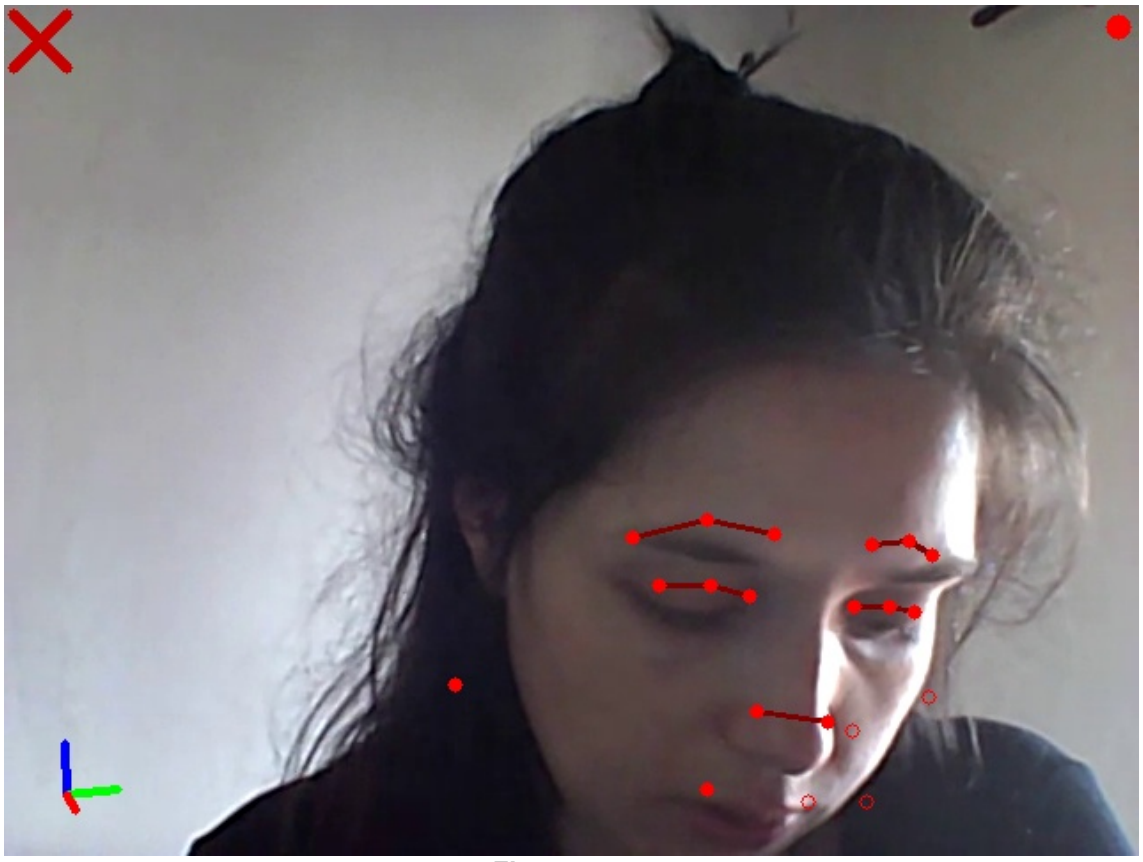


Figura 6. Webcam 2

Como podemos ver na Figura 5, todas as saídas foram corretas, porém na Figura 6 os pontos faciais não conseguiram ser todos encontrados pela Rede Neural, pois o queixo da pessoa ficou fora da área da webcam.

O resultado foi bom, pois não tivemos erros quanto a localização da face, que é o principal tema do artigo.

4.2 Análise Youtube

Utilizamos o trailer do filme "Os Vingadores: Guerra Infinita" para a análise de vídeo do youtube.

Na Figura 7 podemos ver que a rede acertou todas as saídas, diferente resultado para a Figura 8 onde erramos todas as saídas. Isso se deu ao fato do treinamento ter sido feito com um *dataset* com imagens de rostos muito próximos.

A conclusão é que essa análise teve um resultado razoável, pois não sabíamos que as imagens com rostos afastados não iriam conseguir acertar nenhum resultado.

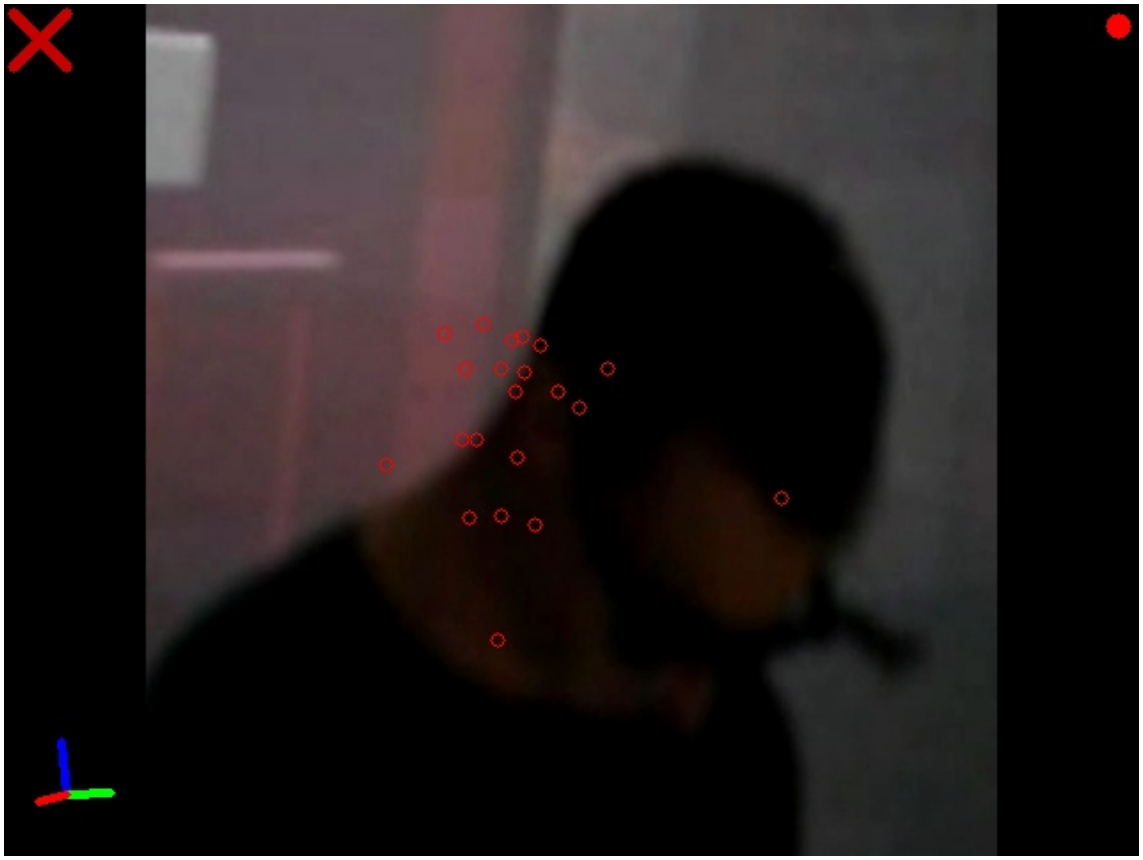
Figura 7. Youtube 1

Figura 8. Youtube 2

4.3 Análise de Salford

Esta análise obtivemos o pior resultado. Como o vídeo passado estava muito afastado, e já sabiamos que a rede não ia conseguir acertar nada. Foi feito um zoom de aproximação para tentar melhorar a acertividade da RNC. Contudo perdemos muita qualidade na imagem e além disso a pessoa gravada estava utilizando um óculos de realidade virtual que atrapalhou bastante.

Na Figura 9 e 10 podemos ver que não tivemos nada de acerto e concluimos que obtivemos um resultado ruim.



igura 9. Salford 1

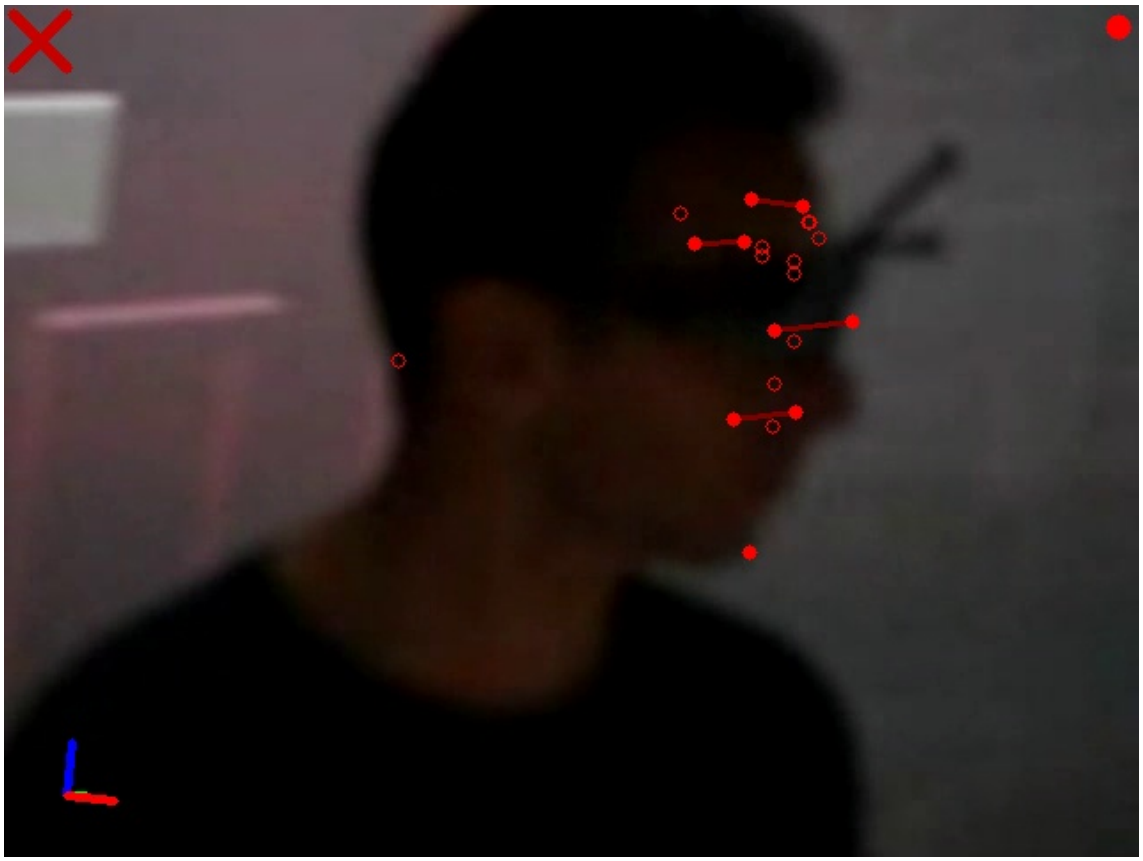


Figura 10. Salford 2

5. Conclusão

Inteligência Artificial é um assunto muito em alta hoje em dia. Fizemos uma pesquisa de redes neurais para analisar a movimentação corporal de uma pessoa restringindo a movimentação facial. Esse assunto pode ser aplicado nas áreas de carros autônomos, segurança e até deficiências motoras.

Acredito que conquistamos um resultado satisfatório de modo geral, porém como podemos ver o *dataset* utilizado para o treinamento faz muita diferença nas análises.

Criando um dataset com os dados de Salford para realizar o treinamento com certeza conquistaríamos resultados melhores para a validação do vídeo. Entretanto como já vimos na seção 3 utilizamos o treinamento supervisionado que foi detalhado em 2.2 e para isso usamos um *dataset* pronto.

6. Trabalhos Futuros

Para trabalhos futuros seria interessante fazer a modificação do código para aceitar mais de um rosto por vídeo e também fazer um treinamento com um dataset que tenha imagens de pessoas que estão distantes da câmera.

Fazendo um pré-processamento dos dados utilizando uma outra rede neural que identificasse um rosto, por exemplo a openface proposta por Baltrusaitis et al. (2018) e utilizando o resultado dela como entrada para esse trabalho obteríamos um resultado melhor.

Outra sugestão seria fazer todo o código utilizando a biblioteca *TensorFlow* ao invés de utilizar a *Chainer v3* que foi utilizada nesse projeto. Porque ao realizar o treinamento no computador local muitas vezes ocorreu exceção de limite de memória mesmo tendo 8gb de ram, então utilizamos um servidor da Amazon c3.x2large para conseguir treinar sem erro. Além disso a comunidade e a documentação do *TensorFlow* é muito mais completa e não tem tantas mudanças de uma versão para a outra como foi o caso do *Chainer* da versão 1 para a 3.

Referências

BALTRUSAITIS, T. el al. OpenFace 2.0: Facial behavior analysis toolkit. In: 2018 13th IEEE International Conference on Automatic Face Gesture Recognition (FG 2018). [S.l: s.n.], 2018. p. 59-66.

BENGIO, Y.; GOODFELLOW, I. J.; COURVILLE, A. Deep learning. Nature, v. 521, p. 436–444, 2015

COPPIN, B. Inteligência Artificial. Rio de Janeiro, Brasil: LTC, 2013. 636 p.

HAYKIN, S. Redes Neurais: princípios e prática. 2. ed. Porto Alegre, Brasil: Bookman, 2001. 900 p.

KOESTINGER PAUL WOHLHART, P. M. R. M.; BISCHOF, H. Annotated Facial Landmarks in the Wild: A Large-scale, Real-world Database for Facial Landmark Localization. In: Proc. First IEEE International Workshop on Benchmarking Facial Image Analysis Technologies. [S.l.: s.n.], 2011.

NGUYEN, D. T.; LI, W.; OGUNBONA, P. O. Human detection from images and videos: a survey. pattern recognition. p. 51:148–175, 2016.

RANJAN, R.; PATEL, V. M.; CHELLAPPA, R. Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition. arXiv preprint arXiv:1603.01249, 2016

RUSSEL, S.; NORVIG, P. Inteligência Artificial. Rio de Janeiro, Brasil: ed. Elsevier, 2004. 1021 p.

WANG, H. Detection of Humans in Video Streams Using Convolutional Neural Networks. 2017.