Universidade do Minho

# Ambiente virtual com deteção de movimentos humanos

## Laboratório em Engenharia Informática


| Guilherme Viveiros | Luís Macedo | Guilherme Andrade |
|---|---|---|
| A80524 | A80494 | A80426 |
| MIEI | MIEI | MIEI |


Julho 2020

# Resumo

Este documento consiste no relatório correspondente ao trabalho prático realizado no âmbito da Unidade Curricular Laboratório de Engenharia Informática, do curso de Mestrado Integrado em Engenharia Informática da Universidade do Minho, no ano letivo de 2020/2021.

É primeiramente fundamentada a construção do ambiente tendo em conta a sua utilidade e viabilidade, elaborando-se também um plano para o seu desenvolvimento.

O processo de especificação do sistema é em seguida pormenorizado, apresentando-se com particular detalhe a implementação das sucessivas etapas para a construção do ambiente virtual, sendo que cada etapa tem a sua fundamentação e objetivos, tentando assim dar a entender as sucessivas tarefas na construção deste específico ambiente virtual.

Seguidamente, apresentam-se vários processos de otimização, não efetuados neste protótipo, mas que constituem um fator fundamental na continuação do sistema, em termos de *performance* e eficiência, caso este, seja decidido ser lançado para o mercado.

**Palavras-Chave:** *Gestual Commands*; *Computer Vision*; *Machine Learning*; *Deep Learning*; *Neural Networks*; *Virtual Environment*; *Desktop task simulator*

# Índice

# Índice de Figuras

# 1. Introdução

Este relatório documenta o trabalho desenvolvido no âmbito da Unidade Curricular de Laboratórios de Engenharia Informática, do curso de Mestrado Integrado em Engenharia Informática da Universidade do Minho, no ano letivo de 2020/2021.

## 1.1. Contextualização e caso de estudo

Ao longo dos anos, independentemente do sector de actividade em questão, o desenvolvimento maioritário de aplicações deve-se ao conforto e conformabilidade do utilizador.

Uma das características mais relevantes de uma aplicação, para além do que fornece, é a sua simplicidade. Nos tempos decorrentes esta característica está internamente ligada com tarefas autónomas que por sua vez entram no ramo de Inteligência Artificial e *Machine Learning*.

Através da utilização desta área, do poder de computação e capacidade de armazenamento existentes apresenta-se aqui, um novo ambiente que não está "à distância de um *click*", mas sim, à distância de um gesto, sem necessidade de interação física entre homem e máquina.

Este ambiente, introduz uma nova estratégia que tem potencial para mudar a maneira que as operações, que serão aqui descritas, têm vindo a ser utilizadas, seja por *hotkeys*, por rato, ou por qualquer outro aparelho externo, todo este processo pode ser substituído por interações gestuais.

A utilidade do ambiente virtual providenciado está inteiramente dependente do utilizador, é este que determina como o pode utilizar para seu benefício e conforto. Mas são vários os casos em que este método pode ser útil, respetivamente:

- O utilizador está cozinhar, seguindo uma receita no seu portátil. Tendo as mãos "sujas", este vai ter trabalho extra (limpar/secar as mãos) para que posso executar alguma ação no seu portátil sem o sujar.
- O utilizador pode estar a utilizar monitores externos e o computador estar longe.
- Caso o utilizador não tenha um aparelho externo, como o rato, que o ajude nas tarefas aqui demonstradas, e que não sabia fazer de outra forma.

Sendo este apenas um protótipo a modelar, utilizam-se apenas algumas operações para a construção do mesmo, respetivamente:

- *Zoom in*, aumento de uma página;
- *Minimize*, minimização de uma página;
- *Swipping left*, passar para o ambiente de trabalho da direita.

.

## 1.2. Objetivos

Objetiva-se com a construção deste ambiente:

- Desenvolvimento de um *pipeline* de redes neuronais que permita a deteção do movimento em vários ambientes e que posteriormente determine qual a operação que o utilizador pretende na sequência do movimento.
- Captar em tempo real o utilizador, processando também em tempo real todas essas *frames*.
- Conseguir executar a operação determinada anteriormente no respetivo sistema operativo.

# 2. Deteção e reconhecimento do gesto

Esta secção tem como presente efeito explicar a deteção do movimento através de um *pipeline* de redes neuronais. Podemos pensar neste *pipeline* através da analogia de um assistente pessoal, como a **Alexa**[1] ou a **Siri**[10], em que primeiramente ocorre a deteção de uma palavra específica, neste caso, "*Hey Siri*" e "*Alexa*", e sucessivamente ocorre o processamento do texto. Da mesma forma, o nosso ambiente dá *trigger* quando deteta uma mão com uma determinada confiança e de seguida processa o determinado movimento sendo que este, pode ou não, ser um gesto a ser executado pelo sistema operativo.

## 2.1. Redes Neuronais

Dado que as redes neuronais são os únicos algoritmos utilizados neste projeto, esta secção tem como objetivo uma sucinta explicação das mesmas, a origem de redes profundas e o porquê do uso destas.

É de conhecimento comum, que as redes neuronais são algoritmos muito poderosos e usados nos mais vastos problemas. Isto porque, em termos simples, uma rede neuronal pode ser descrita como um conjunto de funções normalmente não lineares que têm o objetivo de modelar um função do tipo $Y \approx g(X)$, como podemos observar pela figura 1.

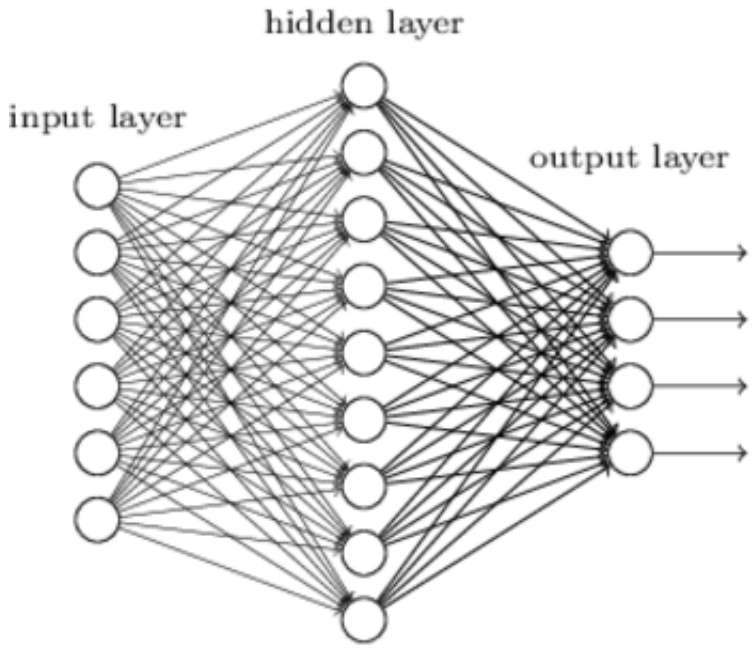



Figure 1: Rede Neuronal

Neste caso, a primeira camada representa os *inputs(X)* e a última camada representa os *outputs(Y)*, e sendo este um problema de aprendizagem supervisionada, ambos são fornecidos à rede tendo esta como objetivo modelar uma função **g**, que é representada pelos pesos na camada intermédia. Por norma, quanto maior é a complexidade e dimensão do problema mais camadas intermédias estão contidas numa rede neuronal, daí a origem de **Deep Learning**, redes neuronais profundas, como pode ser observado na seguinte figura 2.

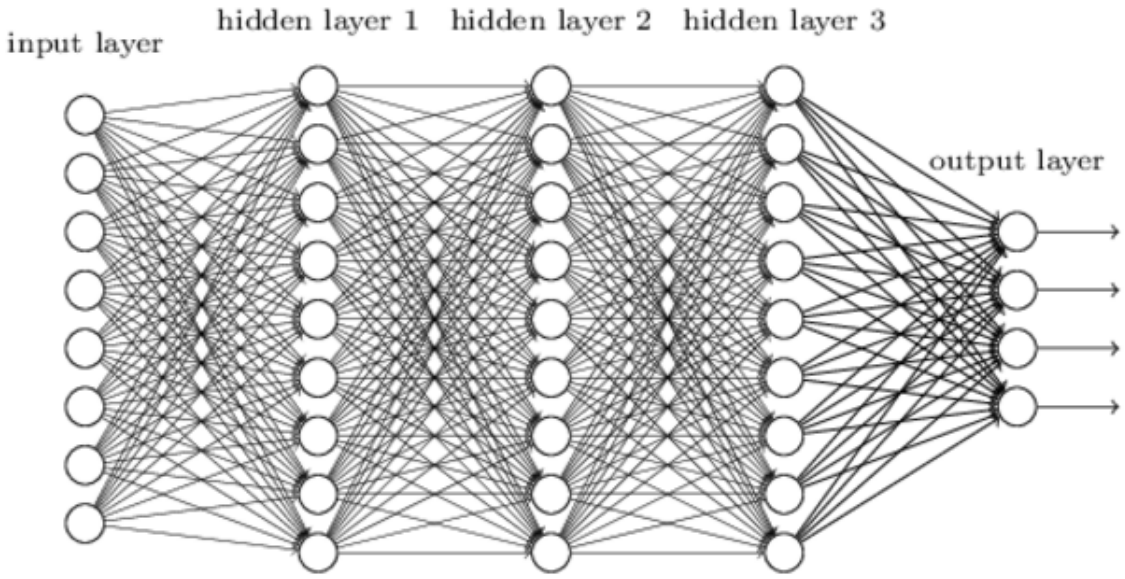



Figure 2: Rede Neuronal Profunda

Para além do tipo de arquitetura aqui apresentado, existem outras, como redes convolucionais[2] e redes recorrentes[9], cada uma apropriada para uma determinada tarefa.

As redes convolucionais serão as que vão ser aqui utilizadas pois são as mais apropriadas para classificação de imagens, pois conseguem preservar as relações entre os pixeis de uma imagem ao aprender as *features* adequadas usando pequenos quadrados, conhecidos como filtros, no *input*, como se pode observar na figura 3. Para além disso, dado que vamos trabalhar com imensos conjuntos de dados e problemas complexos esta é sem dúvida a melhor escolha.

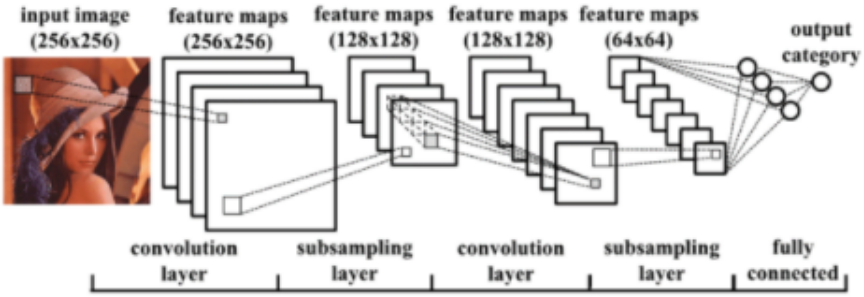



Figure 3: Rede Convolucional

## 2.2. Trigger Hand

Como a maioria das tarefas de **Deep Learning**, a parte mais expansiva e trabalhosa é a criação ou a procura de um conjunto de dados apropriados à tarefa. Dado a pequena dimensão do nosso grupo e dado o facto que o performance em tarefas de **deep learning** estar maioritariamente associado ao vasto conjunto de dados optou-se pela segunda opção, a procura de um *dataset*.

Esta tarefa não foi trivial. De facto, a maneira como se decidiu enquadrar o problema não ajudou no encontro de um *dataset*. Convém realçar que o *trigger hand* aqui descrito não é um problema de classificação binária, em que apenas se deteta se a mão está ou não presente, mas sim um problema de *bounding boxes*, em que, não só a mão tem que ser detectada como a sua posição na imagem completa tem que ser fornecida.

Esclarece-se antes de mais, que o problema apesar de mais complexo e pesado foi enquadrado desta forma, pois pretende-se continuar este protótipo com as mais diversas e distintas operações que seja possível simular no sistema operativo através de um gesto, como fazer *scroll* contínuo, clicar numa determinada operação numa página web, abrir um determinado programa, todos estes necessitam de coordenadas espaciais, providencia-das pelo método proposto.

Dito isto, este *dataset* teve que ser o mais semelhante ao ambiente em que este sistema irá operar, neste caso, para utilizadores que estejam à frente de um computador e em que a qualidade da câmara, luminosidade e ruído adjacentes possam variar inumeradamente.

Experimentou-se primeiro com o conjunto de dados fornecido pela **Oxford**, **Hands Dataset**[8], mas o resultado não foi muito agradável. Dado a escassa pesquisa e disponibilidade presente nesta área, *movement recognition for desktop*(sem a utilizações de sensores), recorreu-se a um *dataset* não muito apropriado aos termos descritos anteriormente como podemos observar pela figura 4, mas em que os resultados obtidos são satisfatórios, e suficientes, para mostrar a eficiência deste protótipo, respetivamente, **Egohands Dataset**[4].

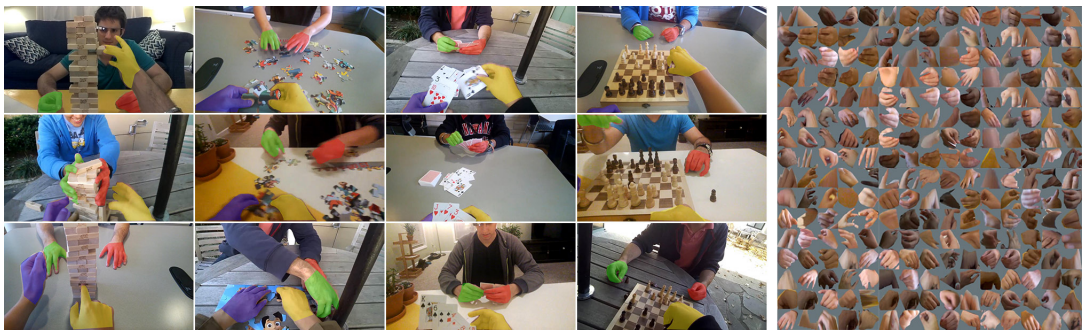


Figure 4: **Egohands Dataset**

Este *dataset* serviu bem por várias razões, para além de conter imensos dados com 15,053 *labels hands*(neste caso com as *bounding boxes*), contém alta qualidade de imagem e a possibilidade de distinguir entre as mãos do utilizador e de pessoas adjacentes, tal como a distinção entre mão esquerda e direita, que para o nosso ambiente é fundamental. Para além disso, fornecem um vasto conjunto de espaços distintos onde estes fazem as mais diversas atividades, fornecendo assim um *dataset* mais complexo e robusto.

### 2.2.1. Preparação do conjunto de dados

Decidiu-se utilizar o **tensorflow** como biblioteca para todos os métodos utilizados no processo de deteção da mão. Tendo o conjunto de dados pronto para utilizar, elaborou-se um

pequeno *script* em **Python** para a transformação deste conjunto em **tfrecords**, que é o que o **Tensorflow** utiliza para treinar os modelos.

Decidiu-se utilizar 80% dos dados como conjunto de treino, 10% para validação e os restantes 10% para teste, cirando desta forma 3 **tfrecords**.

### 2.2.2. Modelo utilizado

Após o tratamento e preparação do *dataset*, a próxima tarefa, é o treinamento do modelo no mesmo. Utilizou-se redes neuronais para este processo, e para estas, é possível utilizar um processo designado de *Transfer Learning* para diminuir o tempo necessário para treinar o modelo. Isto significa que podemos utilizar um modelo existente, que já foi treinado bem numa área ou situação semelhante (neste caso de deteção em imagens) e modificar os últimos *layers* a nosso benefício.

Dado que as redes neuronais normalmente contêm milhões de parâmetros e que o processo de treino pode levar dias, semanas ou até meses a treinar, **Transfer Learning** ajuda a diminuir o tempo para possíveis horas.

Para além disso, na escolha do modelo tem-se em atenção à performance do nosso ambiente, ou seja, voltando ao exemplo da **Siri**, sendo esta uma boa analogia, este modelo está sempre à espera da *trigger word*, "*Hey Siri*", sendo que em todo o momento está a processar o que recebe. O modelo aqui descrito tem o mesmo destino, todas as *frames* captadas são analisadas e potencias *triggers*.

Dito isto, procurou-se um modelo já treinado nesta área e que seja o mais rápido possível. Felizmente o **Tensorflow** já oferece alguns modelos, e decidiu-se utilizar o modelo **ssd_mobilenet_v1_coco** como ponto de partida dado ser um dos modelos atuais mais rápidos para este tipo de tarefa[6].

### 2.2.3. Plataforma utilizada para treino do modelo

O treino do modelo pode ser realizado localmente, em *GPUs* locais que provavelmente irão demorar mais do que na *cloud*, sendo que optamos por usar a *cloud*.

Apenas por referência, o treino do modelo num **MacBooks Air** (intel i5 2.6GHZ, 8GB RAM) pertencente a um dos elementos do grupo, ocorre a uma velocidade máxima de 8 segundos por segundo com *CPU* e 3 segundos com *GPU*. Iria demorar cerca de 17 dias para correr aproximadamente 200 mil épocas no *Mac*, referido anteriormente, comparado com 3 a 4 horas na *cloud*.

Dito isto, utilizou-se o **Google Colab** como plataforma, não é a melhor escolha relativamente à **Google Cloud**, mas é grátis, sendo este um ponto forte na nossa decisão. O **Google Colab** varia o *GPU* de utilização em utilização pelo que não se consegue referir qual o *GPU* utilizado.

Como todo este modelo e sucessiva *API* é fornecida pelo **Tensorflow**, utilizou-se o **Tensorboard** na visualização dos modelos, sendo o parâmetro mais significativo a *loss* do mesmo no conjunto de dados de teste. Dado a utilização da plataforma **TensorFlow Object Detection API** para deteção de imagens, os gráficos de treino não foram disponibilizados, sendo que apenas será exemplificado a *loss* do modelo no *test set*.

Deixou-se o modelo a correr em 200 mil épocas e o resultado pode ser observado na seguinte figura 5 em que o gráfico apresentado contêm um *smooth* de 0.5.

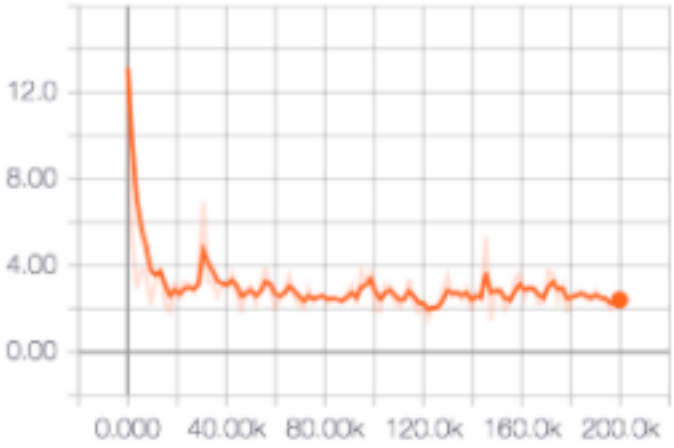
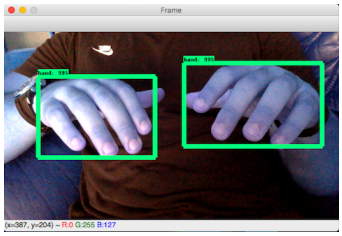


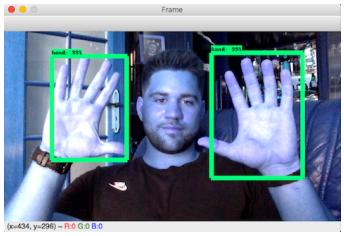

Figure 5: *Loss* do modelo

Após o treino estar completo é criado um *inference graph* (entre outros ficheiros) que de seguida são exportados para todos os colaboradores para que consigam efetuar a deteção da mão.

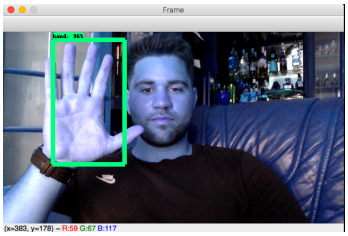
Após a conclusão do modelo, adequou-se o mesmo para a captura de *frames* em tempo real e o resultado pode ser observado nas seguintes figuras.




((a)) Deteção 1




((b)) Deteção 2




((c)) Deteção 3

Figure 6: Modelo de deteção de mãos

## 2.3. *Movement Detection*

Uma parte vital de todo o projeto é o reconhecimento da operação a executar, demonstrada e realizada por um utilizador a tempo real. Este reconhecimento será feito por uma rede neuronal treinada, incorporada no sistema.

O primeiro passo prendeu-se, mais uma vez, à procura do conjunto de dados, previamente processado e tratado, que pudesse ser utilizado. Após alguma procura e consulta, encontrou-se o **20BN-JESTER**[5], uma grande coleção de excertos de vídeos de vários indivíduos distintos a executar gestos predefinidos, em que cada excerto se traduz por um conjunto de *frames*.

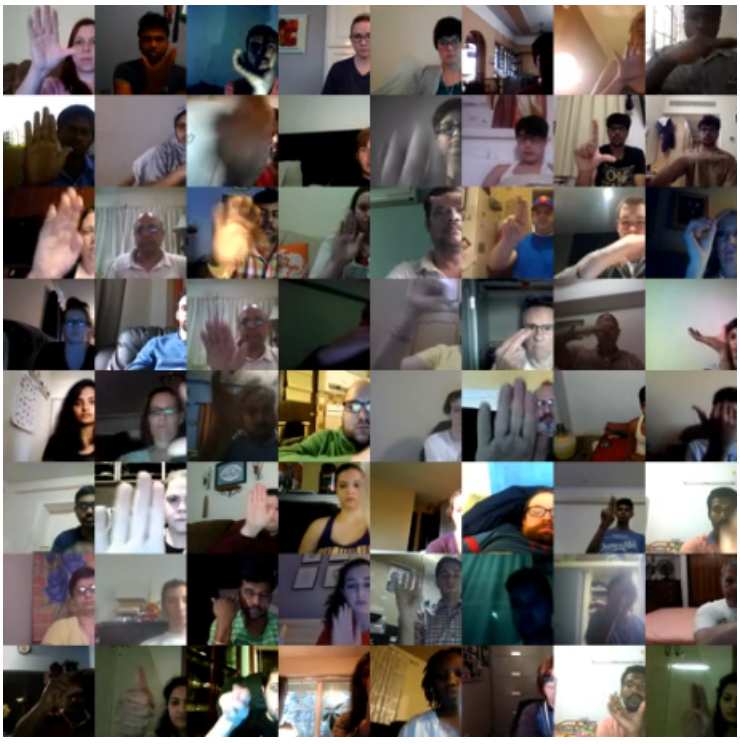


Figure 7: Amostras de *frames* do *20BN-JESTER*

Um excerto seria composto em média por 36 *frames*, consequentes de um processamento ao vídeo original a partir do qual se extraiu 12 *frames* por segundo, o que implicaria que qualquer gesto tem uma janela temporal de 3 segundos para ser executado, apesar de que a

maioria das *frames* são movimentos transitivos para início ou fim do gesto, o gesto em si tem uma presença relativamente reduzida, em termos do número de *frames* como se pode observar pela figura 8.

Figure 8: Sequência de *frames* do gesto *zooming*

Ao verificar que os gestos que se pretendiam reconhecer no projeto se encontravam também nesta coleção, optou-se pela utilização deste *dataset* como base. Então, do conjunto de dados inteiros, usar-se-iam os *clips* correspondentes a quatro classes diferentes: **swiping left, swiping down, zooming in with full hand** e **doing other things**, esta última classe é aplicada quando nenhum do gestos é reconhecido.

De seguida, a ideia passou para a criação de um *dataset* pessoal à imagem deste, onde os membros do grupo procederiam a manter algumas características do *dataset* base estabelecido, como a janela de execução e mais importante ainda a velocidade de execução.

Os vídeo pessoais foram processados da mesma forma, no que toca ao número de *frames* (12 por segundo), no entanto foi adicionada uma característica que permitiu aumentar exponencialmente o tamanho do conjunto de dados: cada gesto executado no vídeo teria 4 segundos de ações de transição entre eles, o que permitia que, para uma mesma instância de um mesmo gesto, este pudesse ser apresentado algures no início, meio ou fim dos 3 segundos estabelecidos pelo **20BN-JESTER**; foi feito um *script* ao estilo de uma janela deslizante que, tendo assinalado a primeira e útlima *frame* exata correspondente, respetivamente, ao início e fim do

gesto (não da janela de 3 segundos mas sim do gesto em si), faria um *sampling* sequencial que colocaria este exato intervalo nas posições possíveis ao longo do tempo de vídeo de 3 segundos. Isto seria útil pois, para além de aumentar o número de dados, neste caso sendo um processo de *data agumentation*[3], um gesto poderia ser teoricamente detetado corretamente se colocado em qualquer posição do *clip*.

## 2.4. Processamento dos dados

Dado a enorme dimensão do conjunto de dados, respetivamente a junção do **20BN-JESTER** e o *dataset* criado manualmente, que como um todo ocupavam entre os 20GB utilizou-se **Out-of-core Training**.

**Out-of-core Training**[7] é um método utilizado quando é praticamente impossível treinar os modelos no *dataset* inteiro, dado não haver memória suficiente na máquina, entre outros casos, sendo que esta forma, possibilita, que os dados estejam disponíveis de uma forma sequencial a cada passo.

Dito isto, criou-se um gerador customizado, também com a *API* do **keras** de forma a mandar **X** em **X** gestos para a memória, em que **X** representa o *batch-size* predefinido. Como foi mencionado anteriormente, o **20BN-JESTER** contém gestos que podem ser inferiores ou superiores a 36 *frames*, logo, decidiu-se defenir o limite máximo de *frames* para 36, correspondentes a 3 segundos, e caso algum gesto contivesse menos que isso, era efetuado um *masking*, caso fosse superior era cortado para as 36 *frames*. Para além disso todas as *frames* são reduzidas para (100,250) pixeis.

### 2.4.1. Modelo utilizado

O próximo passo seria a criação e *design* da arquitetura a usar para a rede convolucional. Neste passo foi ainda considerada a possibilidade de utilizar modelos pré-treinados, mas optou-se por um modelo original de forma a poder conter características específicas dos dados originais. A arquitetura desenhada foi inspirada numa já existente, **C3D**[11], que uti-

liza várias camadas de convolucionais 3D, *pooling* e densas para retirar atributos espaciais e temporais de cada imagem. A arquitetura original apresenta uma complexidade considerável, o que levou à decisão de usar uma aproximação relativamente mais simples para o problema em questão. A arquitetura do modelo pode ser observado na figura 11.

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv1 (Conv3D) | (None, 18, 38, 50, 16) | 1312 |
| pool1 (MaxPooling3D) | (None, 18, 19, 25, 16) | 0 |
| batch_normalization_13 (Batc | (None, 18, 19, 25, 16) | 64 |
| dropout_13 (Dropout) | (None, 18, 19, 25, 16) | 0 |
| conv2 (Conv3D) | (None, 18, 19, 25, 32) | 13856 |
| pool2 (MaxPooling3D) | (None, 9, 9, 12, 32) | 0 |
| batch_normalization_14 (Batc | (None, 9, 9, 12, 32) | 128 |
| dropout_14 (Dropout) | (None, 9, 9, 12, 32) | 0 |
| conv3a (Conv3D) | (None, 9, 9, 12, 64) | 55360 |
| conv3b (Conv3D) | (None, 9, 9, 12, 64) | 110656 |
| pool3 (MaxPooling3D) | (None, 4, 4, 6, 64) | 0 |
| batch_normalization_15 (Batc | (None, 4, 4, 6, 64) | 256 |
| dropout_15 (Dropout) | (None, 4, 4, 6, 64) | 0 |
| conv4a (Conv3D) | (None, 4, 4, 6, 128) | 221312 |
| conv4b (Conv3D) | (None, 4, 4, 6, 128) | 442496 |
| pool4 (MaxPooling3D) | (None, 2, 2, 3, 128) | 0 |
| batch_normalization_16 (Batc | (None, 2, 2, 3, 128) | 512 |
| dropout_16 (Dropout) | (None, 2, 2, 3, 128) | 0 |
| zeropad5 (ZeroPadding3D) | (None, 2, 3, 4, 128) | 0 |
| pool5 (MaxPooling3D) | (None, 1, 1, 2, 128) | 0 |
| flatten_4 (Flatten) | (None, 256) | 0 |
| fc8 (Dense) | (None, 4) | 1028 |

Total params: 846,980
Trainable params: 846,500
Non-trainable params: 480

Figure 9: Arquitetura do modelo

Tentou-se sempre utilizar uma arquitetura com poucos parâmetros, pois pretende-se que esta seja utilizada em tempo real. A simplificação desde o modelo original baseou-se primariamente na diminuição de unidades em cada *layer*, mantendo a proporcionalidade entre os novos números. É importante notar que o modelo apresentado não foi o inicial, a redução de unidades foi gradual. O que se procurou foi uma diminuição da complexidade do modelo sem comprometer os valores de precisão apontados pelo modelo, valores estes que se mantiveram constantes até à ordem dos oitocentos mil parâmetros.

### 2.4.2. Plataforma utilizada e treino do modelo

A máquina utilizada foi fornecida pela **Universidade do Minho**, pelo **Departamento de Informática** (DI), que por referência, foi um *GPU Nvidia Quadro P6000*, com 24 Gb de memória e com uma *Ram size* de 62,8 Gb, que permitiu, um *tunning* em poucos dias, daquele que seria o modelo final.

Após passar o conjunto de dados para o respetivo servidor, em poucas horas foi possível obter resultados e analisar de modo a proceder a potenciais alterações ao modelo.

Para o treino do modelo, foi usada a plataforma do **Keras**, caracterizado pelas suas *API*s de alto nível que permitem a produção de modelos a um ritmo elevado.

A arquitetura demonstrada foi treinada com o *optimizer* **SGD**(do **keras**) com *momentum* e com um *decay* do *learning rate*. É treinada durante 50 *epochs*, no qual se usariam *callbacks* para guardar apenas os melhores modelos em termos de *accuracy* do *validation set*.

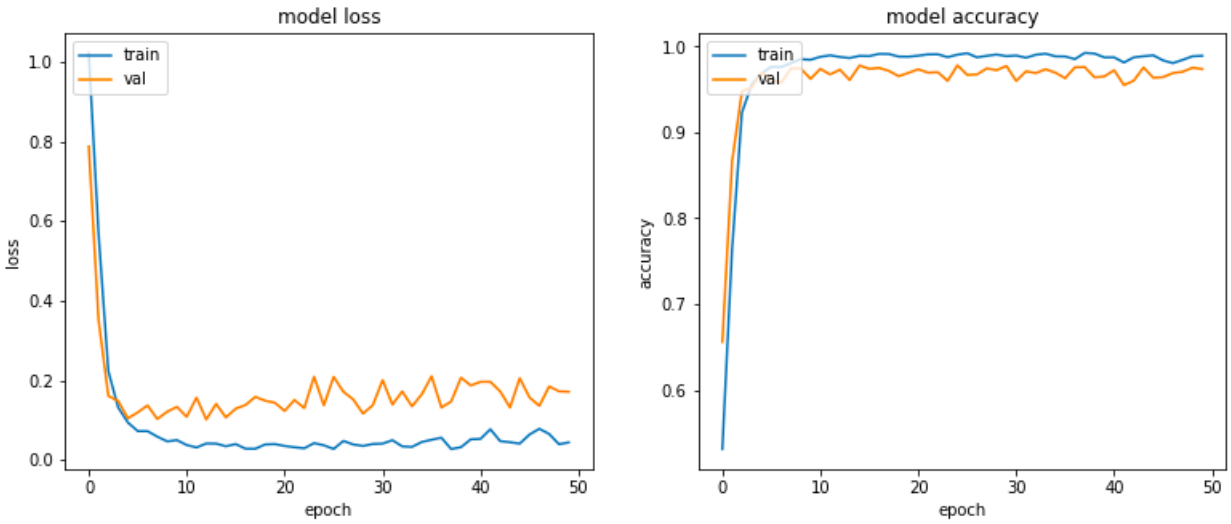



Figure 10: *Accuracy* e *Loss* do modelo

Ao observar as curvas de aprendizagem, verifica-se que, rapidamente (em relação ao número total de *epochs*), atingem-se valores de *accuracy* satisfatórios dentro das primeiras dezenas de iterações ao *dataset*. A *callback* utilizada para guardar apenas os melhores modelos gravou, por último, o modelo da *epoch* número 13, o que indica que a partir da segunda dezena de *epochs* deixa de ser eficiente a continuação do treino, sendo que o modelo final atinge uma *accuracy* no *validation set* de 96% e no *training set* de 98% para deteção de gestos.

# 3. Captura de movimento em tempo real

Por forma a identificar o gesto efetuado pelos utilizadores, é necessário capturar os mesmos em tempo real. Essa captura é feita graças à biblioteca **OpenCV**, que permite aceder a qualquer câmara ligada ao computador.

A utilização do **OpenCV** implica tirar o máximo número de *frames* que um determinado computador consegue tirar por segundo, também conhecido como *fps*. De forma a contrariar este ato pré-defenido pela biblioteca, pois apenas queremos 12 *frames* por segundo de forma a manter a coerência com o *dataset* utilizado, formou-se a seguinte arquitetura:


Figure 11: Alocamento de frames

Cada quadrado representado na figura 11, representa 1 segundo e como pode ser observado, o primeiro quadrado é o passado, o segundo o presente e o terceiro o futuro, desta forma consegue-se obter um *array* temporal com as *frames* em tempo real do utilizador.

Neste caso o primeiro quadrado contém as **X** *frames* passadas, o segundo contém as **X** *frames* atuais e o terceiro quadrado as **X** *frames* que ainda estão para vir no próximo segundo, sendo que **X** está dependente do número de *fps* do computador onde se efetua o processamento.

A vantagem deste tipo de arquitetura é que o número de *frames* pretendido pode ser alterado a qualquer instante, neste caso apenas se pretendem 12 daquelas a que o computador consegue retirar num segundo. Para isto utiliza-se *strides* dinâmicos na escolha das sucessivas *frames*.

Para além disso, o modelo de *trigger word* encaixa-se na perfeição na seguinte arquitetura, pois, quando a mão é detectada, automaticamente o utilizador está no segundo quadrado, já

estão presentes as 12 *frames* anteriores e o utilizador apenas tem 1 segundo para executar o gesto, que após vários testes foi considerado suficiente. Para todos os efeitos estas *frames* também podem ser modificadas, ou seja, é possível ter mais tempo para efetuar o gesto, porém, aumenta a inconsistência perante o *dataset* aqui utilizado.

Estando já esclarecida a parte de captura e alocamento de *frames* em tempo real, especifica-se a modelagem utilizada nos *arrays* descritos anteriormente.

Utilizam-se **Threads** (forma de um processo dividir a si mesmo em duas ou mais tarefas que podem ser executadas concorrencialmente) e *locks*(variáveis associadas a dados que determinam se podem ser efetuadas operações) para organizar o acesso aos recursos, neste caso ao *array* temporal utilizado para alocamento das *frames*.

No total existem 3 **Thread**:

- **Thread** principal, que trata de criar as outras **Threads** e executa o processo principal, neste caso, inicializar o modelo de deteção de mão e gesto;
- **Thread** de captura. Esta **Thread** acede à câmara do portátil do utilizador e captura imagens, que envia para a **Thread** de alocamento de *frames*.
- **Thread** de alocamento de *frames*, que as aloca no respetivo *array* e as pré-processa para estarem de acordo com os dados utilizados nos treinos dos modelos.

A **Thread** de captura, após capturar uma imagem, sinaliza a **Thread** principal para verificar se a imagem capturada contêm uma mão, caso não contenha o processo continua, neste caso a **Thread** continua a angariar *frames*, caso contenha, a **Thread** de alocamento de *frames* espera pelas seguintes *frames* que ainda estão por vir do segundo que falta e de seguida a *Thread* principal utiliza o modelo para detetar se existe algum gesto contido nas 36 *frames*.

# 4. Execução da operação

Com a rede treinada e pronta a deter mãos e classificar os gestos, só falta executar o processo que o gesto representa (minimizar, *zoom in* ou *swipe left*). Para isso decidiu-se simular *hotkeys*.

A simulação é feita usando *packages* em **Python** que permitem simular o pressionar de teclas do teclado. Logo, só falta descobrir quais as *hotkeys* para executar os processos referentes aos gestos nos diferentes sistemas operativos. Como por exemplo, em **Windows 10** para passar para o ambiente de trabalho da direita (*swipe left*) usa-se a combinação de teclas: **WIN + CTRL + seta para a direita**

Todos os testes feitos por forma a testar as *hotkeys* em **MacOS**, foram executados na versão **MacOS Sierra**.

É importante referir que *hotkeys* diferentes às implementadas podem ser adicionadas facilmente, alterando o código fonte, devido à modalidade do mesmo.

# 5. Otimizações

Este projeto surgiu no contexto de trabalho como uma ideia original sobre o qual haveriam poucas referências. Tendo isto em conta, pretende-se dar continuação ao projeto com certas melhorias que advêm da otimização a partir decisões tomadas na realização do trabalho assim como conceitos e *features* ainda não implementadas.

Ao longo do projeto foram feitas decisões para a construção de um protótipo. Estas decisões tinham em mente uma demonstração prática e eficaz daquilo que seria o objetivo, isto é, compreender e realizar remotamente comandos emitidos por uma entidade humana sem auxílio de quaisquer aparelhos.

A utilização de um *dataset* pré-existente como base conceptual para a definição daquilo que seria o gesto permite um melhor grau de normalização em relação a um conjunto de dados personalizado porque apresenta-se melhor estruturado em termos de quantidade e qualidade de amostras. No entanto esta medida apresenta restrições por si só, pois limita o processo de treino à realidade imposta pelos dados, não definidos pelos autores. Como foi mencionado acima, foi adicionado um conjunto de dados construídos à imagem do *dataset* pré-existente. O objetivo será a criação de um conjunto de dados inteiramente original com as características pretendidas como por exemplo a velocidade do gesto e janela de *frames* em que o gesto é realizado (início, meio ou fim). Note-se que se deverá aproveitar o **20BN-JESTER** para este fim, com alguma manipulação das *frames* existentes, através de estratégias de *sampling* de dados desenhadas e já idealizadas.

A implementação do *pipeline* em si, desenhado de raiz, apresenta características altamente adaptativas para vários tipos de máquinas no que toca a *performance* e *frames per second*, aspectos que mostram grande impacto naquele que foi o sistema desenhado. O **pipeline** foi construído baseado em *arrays* dinâmicos que se complementam, auxiliando na construção de um *background* necessário para o modelo criado. Existe com grande espaço para melhoria desde o ponto de vista de *threading*, eficiência e flexibilidade. Estes aspectos a melhorar incidem principalmente sobre a captura e tratamento de imagens, que se traduz como diferente para

cada máquina. É necessário compilar os vários aspectos que contribuem para a portabilidade e viabilidade daquilo que seria uma aplicação de uso comum.

A escolha de *Python* para a implementação baseou-se da usabilidade e simplicidade que a linguagem apresenta, principalmente no âmbito de **Machine Learning**. A simplificação assim como a compatibilidade com várias plataformas necessárias (como **Tensorflow, Keras, OpenCV**) tornou **Python** na escolha óbvio para uma criação inicial. No entanto, a utilização concorrente, ao nível desta linguagem, visto que se trata de um interpretador, não é a mais precisa. Tendo isto em conta, pretende-se a utilização de compiladores otimizados como Numba ou uma transição para C++.

A implementação de apenas um pequeno conjunto de gestos é uma característica bastante importante, pois permite uma manutenção e *tunning/debugging* mais eficiente para encontrar falhas no *pipeline*. No entanto, à medida que este vai sendo desenvolvido e aprimorado, é imperativo a expansão do vocabulário associado aos comandos interpretáveis pelo modelo. Esta é uma medida a desenvolver juntamente com o conjunto de dados personalizados que permitirá uma melhor imersão naquilo que seria o *motion driven desktop*.

A utilização de um *dataset* mais apropriado para a deteção da mão, ou a construção do mesmo é um passo importante na melhoria do ambiente, visto que este conjunto de dados não apresenta as características mais semelhantes aos casos reais, inviabilizando a *performance* do sistema em determinados casos.

No que toca à execução do processo referente ao gesto efetuado, uma possível otimização é a troca entre o uso de simulação, neste caso pressionar as teclas, por chamadas ao sistema operativo mais diretas, ou seja, em vez de utilizar simulações de teclas, executar diretamente o processo desejado. Esta otimização seria alcançada com a linguagem de programação **C++** e é denominada por *Interface*, ligando a nossa aplicação com o sistema operativo.

# References


[1] *Alexa*. 2019. URL: https://www.amazon.com/b?ie=UTF8&node=17934671011.

[2] *Convolutional Neural NEtworks*. 2020. URL: https://pt.wikipedia.org/wiki/Rede_neural_convolucional.

[3] *Data agumentation, get more data with a simple process*. 2019. URL: https://towardsdatascience.com/data-augmentation-for-deep-learning-4fe21d1a4eb9.

[4] *Egohands Dataset*. 2020. URL: http://vision.soic.indiana.edu/projects/egohands.

[5] *Jester Dataset*. 2020. URL: https://20bn.com/datasets/jester.

[6] Wei Liu1 et al. *SSD: Single Shot MultiBox Detector*. 2015.

[7] *Out-of-core training*. 2020. URL: https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781785887215/2/ch02lvl1sec12/out-of-core-learning.

[8] *Oxford Hand Dataset*. 2020. URL: http://www.robots.ox.ac.uk/~vgg/data/hands.

[9] *Recorrent Neural Networks*. 2017. URL: https://matheusfacure.github.io/2017/09/12/rnn.

[10] *Siri*. 2019. URL: https://www.apple.com/siri.

[11] Du Tran et al. *Learning Spatiotemporal Features with 3D Convolutional Networks*. 2015.