Machine Learning
Sign Language Understanding

Luís Fonseca
89066
DETI
Aveiro, Portugal
luiscdf@ua.pt
55%

Pedro Marques
89069
DETI
Aveiro, Portugal
pedromm@ua.pt
45%

***Abstract*—Este documento tem como objetivo dar a conhecer o processo de seleção de um algoritmo de machine learning adequado ao problema em questão - Reconhecimento de Linguagem Gestual, no contexto da cadeira de Tópicos de Aprendizagem Automática da Universidade de Aveiro no presente ano letivo de 2019/2020.**

***Index Terms*—logistic regression, support vector machines, machine learning, linear kernel,neural networks**

I. Introduction

No presente contexto, trabalha-se com o tema proposto 1 - ”Identification of digits from sign language images”, cujo dataset se encontra no site kaggle [6]. Será explicitado o método de estudo adotado e feita uma discussão dos resultados com base em cada método aplicado.

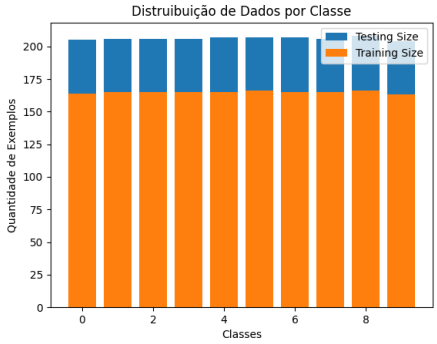
II. Os Dados

A. Estado do Dataset

Primeiramente, é importante referir o facto do dataset apresentado não estar, diga-se ”integro”, apresentando as labels das respetivas imagens fora de ordem. Ora, este problema foi resolvido posteriormente, e a solução para o mesmo encontra-se na mesma página indicada anteriormente.

B. Tamanho dos Dataset

Uma adversidade bastante notória no dataset apresentado, é o facto deste ser relativamente pequeno, em comparação com por exemplo, os de outros temas. Ora, como veremos mais adiante, isto teve um impacto significativo nos resultados obtidos, uma vez que cria uma certa tendencia para overfitting. Para dar uma ideia, o dataset é constituido por cerca de 2062 imagens, distruibuidas por 10 classes. Estando equilibrado, temos cerca de 200 imagens por classe. Com um numero tao reduzido de dados, aquando do testing size, optou-se por escolher 20%, o que se traduz em aproximadamente 160 imagens para treino e 40 para testing - note-se que se garantiu uma divisão equilibrada por classe. Com um número (consideravelmente) maior de dados, eventualmente escolher-se-ia uma percentagem menor, aumentando o volume de treino, o que neste caso não é possível.



III. Metodo Aplicado

Numa primeira fase, e tendo em conta a inexperiencia neste dominio, pensou-se em aplicar todos os metodos e hyperparameters (possíveis) de uma vez, de forma a conseguir filtrar quais os que apresentavam melhores resultados. Contudo não se veio a tornar uma ideia muito brilhante, uma vez que não só era computacionalmente dispendioso, como havia resultados simplesmente inuteis, com accuracies a rondar 0.1. Esta tentativa falhada, não se pode considerar inesperada, contudo foi um primeiro passo interessante, que permitisse uma melhoria do método a aplicar. Numa visão mais empirica, e tendo em conta que datasets mais pequenos encorrem num maior risco de overfitting, quando expostos a modelos mais complexos, decidiu-se portanto priorizar modelos mais simples. Da mesma maneira, considerou-se necessário o uso de cross-validation, uma vez que o mesmo geralmente é útil nestas situações.

IV. Considerações

A. Cost Functions

Para os métodos em que se usou a libraria sklearn, não existe um método que retorne o custo por iteração. Embora existam alguns ”workarounds” na internet, estes nao foram considerados práticos nem ”pythonic”, e por isso descartados. Por esse motivo, e tendo em conta que o dataset esta bastante bem equilibrado, decidiu-se usar a accuracy como métrica principal nos resultados obtidos. Note-se que esta medida por ser falaciosa, principalmente em datasets desilibrados, mas relembra-se que tal não se aplica a este caso.

B. Gráficos Apresentados

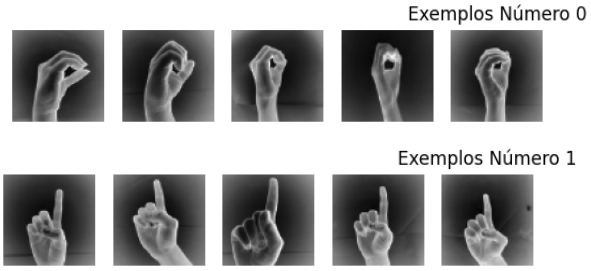
Nos gráficos apresentados, em alguns casos, não se encontram presentes todos os dados estudados durante este projeto. Apenas se incluem os considerados mais relevantes - casos extremos e melhores casos, de modo a minimizar a poluição dos gráficos. Considera-se contudo que esta permissa não afeta negativamente o estudo a ser feito.

C. As Imagens

Como principais caracteristicas do dataset temos:

- tamanho: 64 x 64;
- espaço de cores: cinza;
- formato do ficheiro: npy
- numero de classes: 10
- numero de participantes: 218
- numero de exemplos por participante: 10

Note-se que os dois últimos pontos são aproximações, daí os valores nao corresponderem aos 2062 referidos anteriormente. Em jeito de ilustrativo, aqui se deixam um conjunto de exemplos das imagens presentes no dataset:



V. Metodos

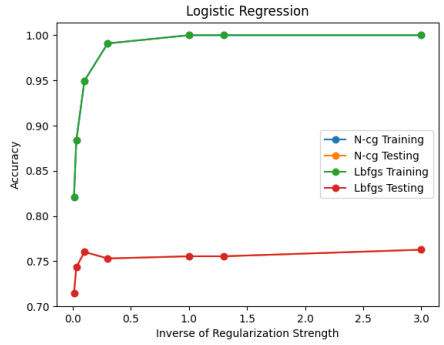
A. Logistic Regression

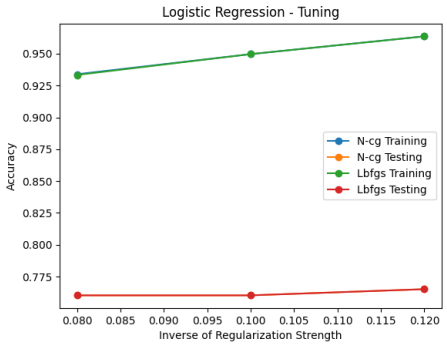
Com o considerado anteriormente, optou-se então por começar com um dos métodos mais simples de machine learning. Graças ao que foi dito no ponto III, nesta etapa foi aplicada uma escolha inicial de hyperparameters mais seletiva e promissora.

1) Parametros: A função da biblioteca sklearn utilizada, LogisticRegressionCV, é como o nome indica uma função encarregada de implementar regressão logistica com validação cruzada. Tal como outras funções usadas mais a frente, apresenta parametros que desconheciamos a posteori, pelo que passaremos agora a uma justificação dos mesmos:

- na generalidade dos casos, pelo desconhecimento em muitos dos parametros, os não mandatórios foram deixados como default;
- como **solver**, escolheu-se *newton-cg* e *lbfgs* - pois eram os metodos mais comuns e são os unicos que suportam "multinomial loss";
- **penalty**: como consequencia da escolha anterior, apenas podiamos escolher penalti "l2" - uma forma de *ridge regression* - o que vai ao encontro do referido anteriormente sobre o problema de overfit, na medida em que esta tecnica é bastante boa a evita-lo;
- **cv**, ou seja o número de k-fold, por esta variavel apresentar um elevado custo computacional - no sentido em que variaçoes deste parametro são dispendiosas de trabalhar com - manteve-se o default de 5. Aplicou-se o método **Stratified K-fold**, que se destaca por tentar equilibrar o numero de exemplos de cada class em cada fold;

2) Resultados: Como se pode ver em ambos os graficos, os dois métodos convergiram para o mesmo valor. Isto deve-se ao facto de que LBFGS é "in a nutshell", analoga ao metodo de Newton, daí ser natural esta sobreposição ao longo do gráfico. A diferença está principalmente no facto de que LBFGS usa aproximações e tecnicas para poupar memória. O que lhe confere potencial em problemas com pequenos datasets [2].

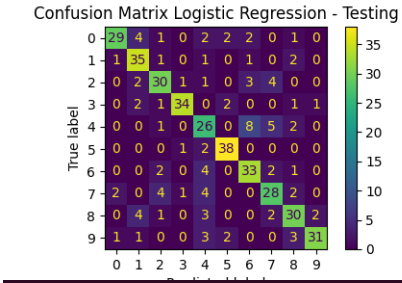




Quanto ao primeiro gráfico, observa-se rapidamente que estamos perante um caso de overfitting, à medida que o C (lembrando que é o inverso de lambda) aumenta. Contudo pode-se apontar o facto de que a amplitude de valores de C é bastante notória, pelo que se decidiu fazer tuning aos valores mais promissores, neste caso à volta do valor C=0.1. Isto leva-nos ao 2o gráfico, que demonstra que de facto era uma suposição correta. Note-se que o ponto C=0.12 apresenta um ligeiro aumento na accuracy, contudo esta é bastante negligenciável para estudo posterior.

Confirmando estas observações, na própria função do sklearn, é possível verificar qual o mehor C para cada classe: [0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1] O mapeamento aqui é feito de forma a que cada index corresponda a uma class. Obtemos então:

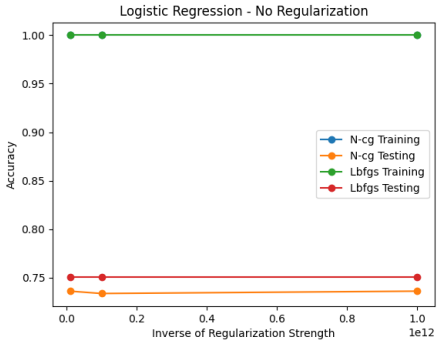
- train accuracy: 0.9496664645239539
- test accuracy: 0.7602905569007264



	precision	recall	f1-score	support
0	0.88	0.71	0.78	41
1	0.73	0.85	0.79	41
2	0.73	0.73	0.73	41
3	0.92	0.83	0.87	41
4	0.57	0.62	0.59	42
5	0.86	0.93	0.89	41
6	0.70	0.79	0.74	42
7	0.68	0.68	0.68	41
8	0.71	0.71	0.71	42
9	0.91	0.76	0.83	41
accuracy			0.76	413

Note-se que cada coluna numerada à esquerda corresponde analogamente à respetiva class.

3) No Regularization: Uma vez que a biblioteca em questão não permite definir regularização nula, a única alternativa é tentar aproximar lambma de zero, aumentando o valor de C o mais possível.



Observando o gráfico em questão é possível verificar que aqui os valores para os dois solvers divergem. Ora, isto parece facilmente justificável pelo referido anteriormente: uma vez que um dos solvers tenta aproximar o outro, ao não haver regularização, estes valores irão naturalmente ter tendencia a divergir. Não obstante em ambos os casos o valor da accuracy diminui, o que se deve obviamente ao facto de o overfitting se tornar mais forte, decido à menor regularização.

4) Conclusão: Sendo um modelo relativamente simples, e tendo em conta que estamos a trabalhar com imagens, ou seja classes com "elevado" numero de features, um modelo linear deste tipo não apresenta uma performance muito elevada nesta situação.

B. SVM

Avançando para um algoritmo mais complexo, deparamo-nos com SVMs. Embora um bom Kernel para começar seja o RBF, devido a sua qualidade "general purpose", decidiu-se começar pelo kernel linear. Recorrendo às ferramentas disponibilizadas pela biblioteca em uso, fez-se uso da função GridSearchCV, que permite uma pesquisa exaustiva por parametros passados, escolhendo por fim os mais apropriados.

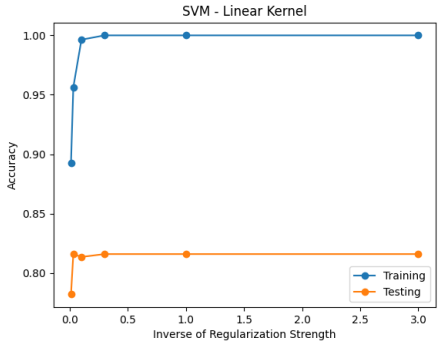
C. Linear Kernel

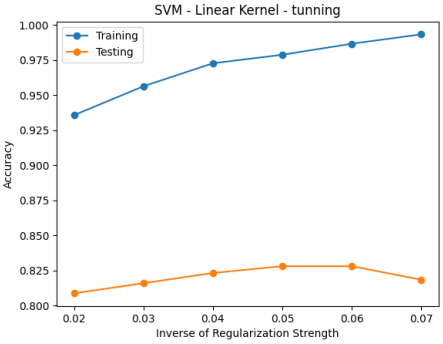
Também conhecido por representar uma SVM sem kernel, parece uma boa alternativa para continuar o processo anterior, tendo em conta que continuamos a trabalhar com modelos lineares.

1) Parametros: Uma vez que não possuí Kernel, não existem muitos Hyperparameters com os quais se possa trabalhar, exceto o valor C, que também aqui representa o inverso da força de regularização (for 1/gamma).

- como **cv**, usado na GridSearchCV, mais uma vez optou-se pelo default, StratifiedKFold 5-fold;

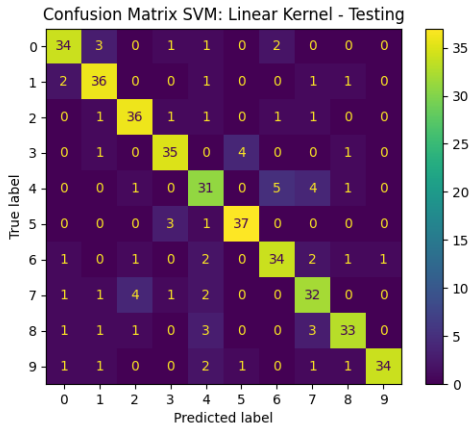
2) Resultados: Relativamente ao algoritmo anterior, este apresenta melhores resultados. Não obstante, com o aumento do C, rapidamente se atingem valores constantes, fazendo que tanto as linhas de treino e teste se tornem literalmente paralelas, tendo-se chegado a um caso de overfitting.





Com isto em conta, realça-se o foco portanto em valores de C mais reduzidos, Pela observaçao do segundo gráfico, é de realçar o tuning que foi feito, e a forma como o incremento de C foi feito meticulosamente de forma a maximizar, dentro dos possíveis, os valores de accuracy. Assim, para C = 0.6, temos

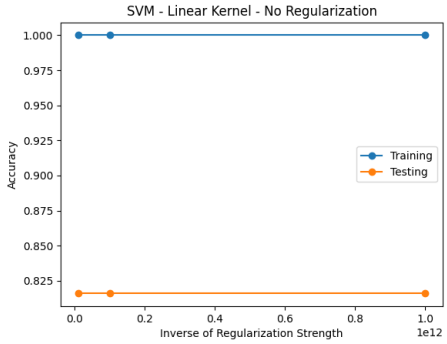
- train accuracy: 0.9866585809581565
- test accuracy: 0.8280871670702179



	precision	recall	f1-score	support
0	0.85	0.83	0.84	41
1	0.82	0.88	0.85	41
2	0.84	0.88	0.86	41
3	0.85	0.85	0.85	41
4	0.70	0.74	0.72	42
5	0.88	0.90	0.89	41
6	0.81	0.81	0.81	42
7	0.73	0.78	0.75	41
8	0.87	0.79	0.82	42
9	0.97	0.83	0.89	41
accuracy			0.83	413

3) No Regularization: É possível observar que ao contrário do caso anterior sem regularização, neste caso não há alterações à medida que C aumenta. Ora, com o intuito de justificar tal facto, há que ter em conta dois fatores que acontecem durante o processo de calculo da SVM:

- procura-se um hyperplano com o maior minima margem;
- procura-se um hyperplano que separe corretamente o maior numero de casos possíveis;

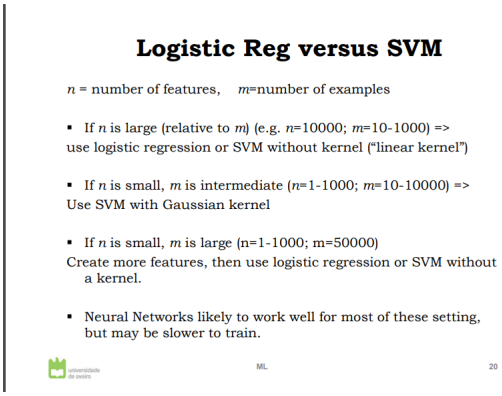


Acontece que nem sempre é possível maximizar os dois, havendo um trade-off entre os mesmos. O parametro C, tem a maior influencia no segundo caso, ou seja, determina o quão separados queremos os casos - na medida em que quanto maior o C, menor a margem permitida. Desta forma, este valor constante com o qual somos deparados, terá origem no facto de que se atingiu um valor minimo C1, a partir do qual não é possível ”separar mais os resultados”, tornando irrelevante qualquer valor seguinte de C [4];

4) Conclusão: Embora inicialmente se pensasse que este modelo e o anterior fossem apresentar resultados muito identicos, devido à natureza linear dos mesmos, tal não aconteceu. Neste caso, a base accuracy subiu consideravelmente. Numa tentativa de possível justificação para tal, relembra-se o facto de que o dataset é bastante pequeno. Ora, nesta situação, **Outliers** têm maior influencia nos dados, o que pode afetar os resultados. Estes Outliers terão mais influencia no modelo anterior, uma vez que o mesmo tem com consideração todas as entradas para o calculo da função de regressão logistica. Por outro lado, no modelo atual, há um enfoque maior na ”decision boundary”, tal que o objetivo passa por colocar esta boundary linear de maneira inteligente. Desta forma, apenas pontos próximos da decision boundary fazem realmente diferença, diminuindo assim a influencia destes outliers, e possivelmente aumento a accuracy do modelo.

D. RBF Kernel

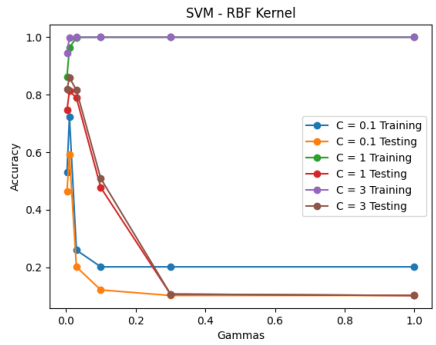
Sigla para (Gaussian) Radial Basis Function, é a escolha mais comum, sendo inclusive a escolha por defeito. Um outro motivo a favor deste e do anterior Kernel encontra-se também no slide 27 da lecture 5

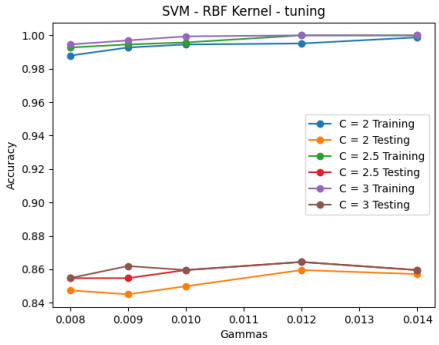


da cadeira de TAA: elaborando, considerou-se que o dataset em estudo se podia encontrar de alguma maneira entre os dois primeiros casos apresentados.

1) HiperParamettros: Neste contexto, a nivel de parametros quer da função de SVM, quer do ”wrapper” GridSearchCV, não foram alterados, mantendo-se os valores default. Quandos aos hiperparametros, este kernel apresenta mais um, gamma, devido à sua natureza Gaussiana. Numa primeira fase foram escolhidos alguns valores arbitrários, mas pensados, para um primeiro desenho dos resultados, ao qual depois foi feito algum tuning nos parametros de forma a maximizar a accuracy.

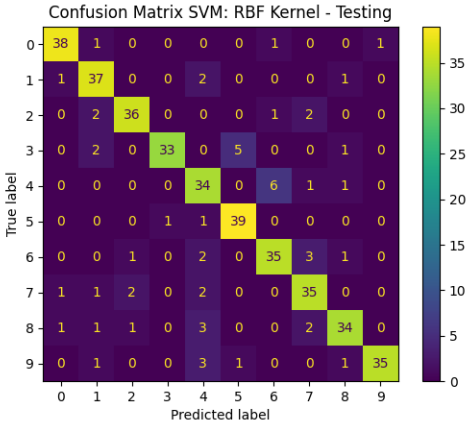
2) Resultados: Neste caso obtemos o melhor resultado, no que diz respeito à accuracy, até ao momento. Isto deve-se ao facto de ser um kernel caracterizado por ter propriedades que o tornam diga-se ”flexivel” para bastantes algoritmos kernel-based.





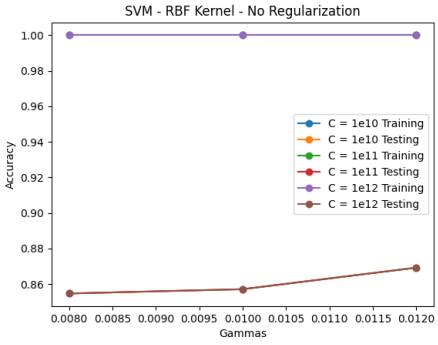
Neste sentido, considerou-se que os melhores resultados seriam obtidos C = 3 e gamma = 0.009:

- train accuracy: 0.9969678593086719
- test accuracy: 0.8619854721549637



	precision	recall	f1-score	support
0	0.93	0.93	0.93	41
1	0.82	0.90	0.86	41
2	0.90	0.88	0.89	41
3	0.97	0.80	0.88	41
4	0.72	0.81	0.76	42
5	0.87	0.95	0.91	41
6	0.81	0.83	0.82	42
7	0.81	0.85	0.83	41
8	0.87	0.81	0.84	42
9	0.97	0.85	0.91	41
accuracy			0.86	413

3) No Regularization: Nesta situação, algo bastante surpreendente aconteceu. Com o aumento de C - ou seja, diminuição da ”força” de regularização, esperava-se que fossem atingidos piores valores de accuracy, ou quanto muito constantes. Não obstante, o oposto aconteceu, e foram observados valores antes não alcançados, embora por uma pequena margem.



Havendo já referido a influencia de C no kernel anterior, surge a necessidade explicar gamma. Gamma define o quao longe um exemplo pode estar para ter influencia noutro. Valores baixos traduzem-se numa ”longa distancia” e valores altos numa ”baixa distancia”. Assim, quando o gamma é muito grande, a area de influencia dos support-vectors inclui apenas o próprio support-vector e a nenhuma regularização pode prevenir overfitting. Por outro lado, quando o gamma é demasiado baixo, não é possível capturar a complexidade do dataset, uma vez que está a ser incluido todo ele. Isto causa um modelo identico ao modelo linear, com hyperplanos que separam centros de alta densidade entre quaisquer 2 classes [3]. No primeiro gráfico, é possível ver o efeito de um valor grande de Gamma, na medida em que o overfitting é notorio, ao verificar como os valores da accuracy de testing baixam com o aumento do mesmo.Por outro lado observa-se também um aumento de accuracy para valores mais baixos e controlados, como se pode ver pelo segundo gráfico. A questão surge portanto no terceiro gráfico, onde para valores altíssimos de C, a accuracy sobre ligeiramente acima do valor máximo visto no gráfico de dois. Concretizando, de 0.8547215496368039 para 0.864406779661017 - no que diz respeiro à accuracy de teste. Ora, esta é uma questão à qual não foi possível encontrar uma resposta plausivel, e que portanto terá de ser uma questão a ficar em aberto.

4) Conclusão: Como referido, atingiram-se os valores mais elevados de accuracy com este algoritmo. Com o Kernel em questão, realça-se o facto do parametro gamma ser de extrema importancia, no sentido em que a variação minima de valores pode causar uma grande diferença de accuracy. Da mesma maneira, sente-se que neste contexto, gamma tem mais influencia na accuracy do que C, baseado principalmente no terceiro gráfico apresentado. Contudo, relativamente a este não se pode concluir muito com base nos resultados, que digam-se são estranhos.

E. Neural Networks

De seguida foram utilizadas Neural Networks para estudar o dataset. Este sistema computacional tem várias aplicações sendo uma dessas a classificação de imagens (por exemplo, diferenciando imagens de cães e gatos).

1) Hyperparameters: Para processar a informação corretamente é preciso ter em mente os hyperparameters utilizados em Neural Networks, como por exemplo:

- Número de Hidden Layers
- Número de Nodes por Layer
- Dropout
- Learning Rate
- Activation Function
- Epochs
- Batch Size

Inicialmente, criou-se Neural Networks com vetores e matrizes utilizando a biblioteca numpy, mas devido a vários obstáculos (falta de experiência, muito tempo para escrever o código manualmente, maus resultados), foi optado por utilizar o Keras, uma API de alto nível para criação, treino e avaliação de Neural Networks. Graças ao Keras, foi possível desenvolver modelos capazes de analisar os dados disponíveis rapidamente.

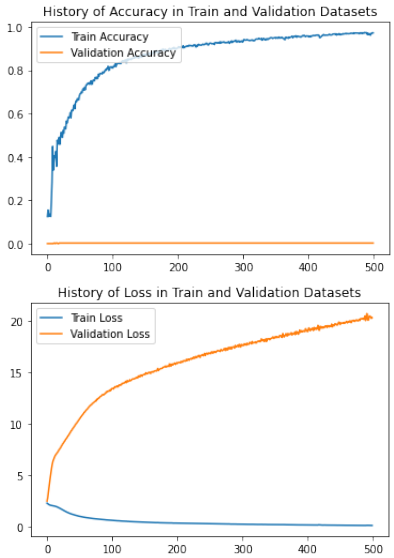
F. Multi-Layer Perceptron

Os primeiros modelos desenvolvidos foram MLP, visto que são simples de implementar e de alterar. Esta classe de Neural Network é composta por múltiplas Layers em que cada node de uma Layer está ligada a todos os nodes da Layer seguinte e a informação é passada de uma Layer para a seguinte de modo feed-forward. O dataset é carregado diretamente dos ficheiros disponibilizados e no caso dos dados de input têm que se alterar a sua shape, visto que a Input Layer necessita de receber informação com uma só dimensão.

1) Arquitetura do Modelo Desenvolvido: Em geral, o formato destas Neural Networks desenvolvidas era:

- 1 Input Layer
- 2 - 4 Hidden Layers (Layers mais proximas da input layer têm maior número de nodes) e com activation function ReLU
- 2 Output Layer, com 10 nodes (1 para cada dígito) e com activation function softmax

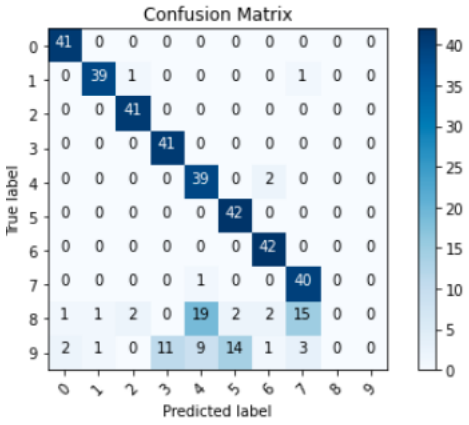
2) Resultados:



Resultados obtidos:

- A Train Accuracy atinge valores superiores a 0.85 mas pelo comportamento do gráfico conclui se que os resultados não são robustos.
- A Train Loss tende para zero
- A Validation Accuracy nunca atinge valores superiores a 0.01
- A Validation Loss cresce a cada epoch

```
Loss 4.610833055729084
Accuracy 0.7759456634521484
```



Conclusão: Qualquer alteração feita aos hyperparameters durante a análise dos dados não alteraram a performance da rede nem o resultado. Estes resultados devem-se fundamentalmente ao facto do datast não ser o mais adequado por causa do seu tamanho reduzido, mas a Neural Network escolhida também não foi a mais adequada. Houve, por isso, necessidade de utilizar um modelo diferente de Neural Network.

G. Convolutional Neural Network

Após vários testes com MLPs foi decidido utilizar Convolutional Neural Networks, visto que estas são eficientes em image classification. Este tipo de Neural Network é constituída por dois tipos de Layers:

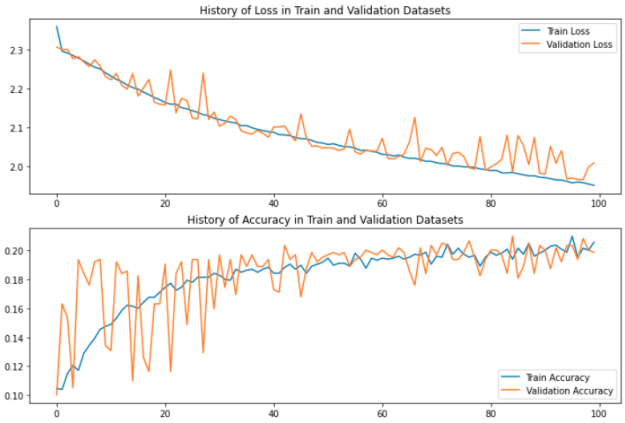
- Convolutional Layers - Reduzem o número de parâmetros de uma imagem através de dimensional reduction
- Dense Layers - Layers semelhantes às de MLP

Ao contrário de MLP, as CNN recebem os dados em forma de ”imagem” e não necessitam de terem alterações na sua shape, mas é preciso dar flatten aos dados que são feed-forward das Convolution Layers para as Dense Layers. De seguida, vai-se explicar como este tipo de Neural Networks foram aplicadas para se atingir resultados melhores em comparação aos anteriormente analisados.

1) Primeiro Modelo CNN: Arquitetura

- 1 Convolution Layer com MaxPooling
- 1 Dense Layer
- 1 Dense Output Layer

Resultados



Analisando os gráficos, verifica-se:

- Training accuracy baixa
- Validation accuracy baixa
- Robustez de validação baixa

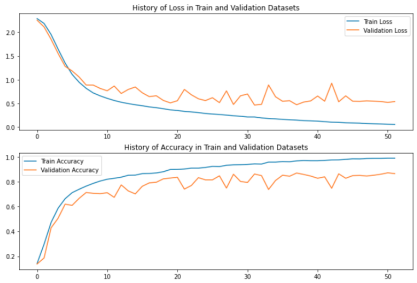
```
Loss: 0.6945900037415669
Accuracy: 0.7917675375938416
```

Para corrigir os problemas deste modelo sera necessário adicionar mais uma Convolution Layer

2) Segundo Modelo CNN: Arquitetura

- 2 Convolution Layers com MaxPooling
- 1 Dense Layer
- 1 Dense Output Layer

Resultados



Analisando os gráficos

- Train accuracy alta
- Validation accuracy baixa
- Robustez de validação é baixa

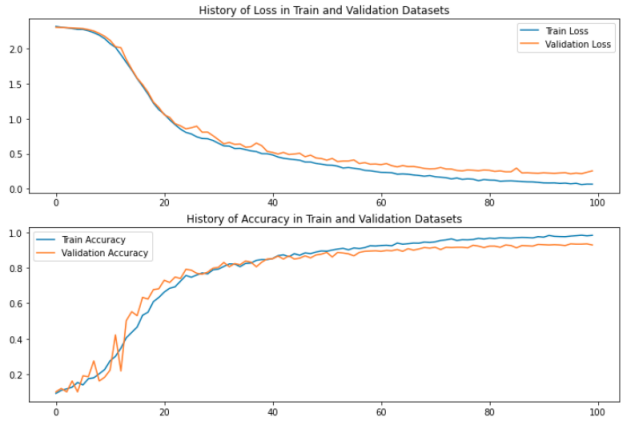
```
Loss: 0.7052285207241558
Accuracy: 0.8595641851425171
```

Para corrigir os problemas que ainda existem vai ser adicionado mais uma Convolution Layer e Dropouts

3) Terceiro Modelo CNN: Arquitetura:

- 3 Convolution Layers com MaxPooling e Dropout
- 1 Dense Layer
- 1 Dense Output Layer

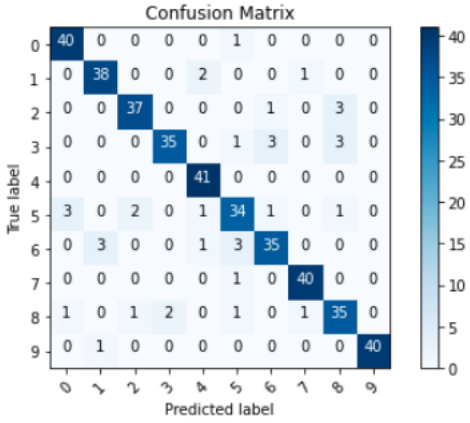
Resultados :



Analisando os gráficos:

- Train accuracy alta
- Validation accuracy alta

```
Loss: 0.7052285207241558
Accuracy: 0.8595641851425171
```



Conclusão: Ao longo do desenvolvimento dos vários modelos de CNN, verifica-se uma melhoria dos resultados obtidos após várias Convolution Layers serem adicionadas e a inclusão de Dropout. Os melhores resultados obtidos devem-se principalmente devido ao facto de Convolutional Neural Networks serem eficientes a trabalhar com imagens, mas os mesmos podiam ser melhores se o dataset fosse maior.

H. Conclusão

Dá-se assim por concluído o estudo deste dataset recorrendo a machine learning. Como pontos importantes que definiram o a execução do projeto, há a referir:

- **experiência** - sendo um conteúdo recentemente lecionado, a pouca experiencia dos autores será de certo um fator a ter em conta
- **tamanho do dataset** - referido algumas vezes ao longo do relatório, este veio a tornar-se uma grande adversidade, que em conjunto com o ponto previamente mencionado, dificultou o trabalho, principalmente quando sempre se procurou atingir uma accuracy de 1 - o que vimos que nao se tornou uma realidade

Relativamente aos algoritmos fora do contexto de NN, estes fizeram uso da libraria sklearn e como foi dito não se tornou possível o desenho de gráficos baseados no custo por iteração. Ao se ter usado a accuracy de test como principal medida de sucesso, tal foi feito com a consciencia que esta não possuí qualquer relação com o custo [5], pelo que tal se salvaguarda aqui. Tendo em mente as otimizações e abstrações próprias desta livraria, acredita-se que é possível contudo, fazer uma certa aproximação entre as duas medidas, no sentido em que também permite o estudo da eficiencia dos algoritmos.

	Logistic Regression	SVM - Linear Kernel	SVM - RBF Kernel	MLP	CNN
Testing Accuracy	0.76029055 69007264	0.82808 716707 02179	0.86198 547215 49637	0.77594 566345 21484	0.90799 0336418 1519

Com isto em conta, finaliza-se o relatório com bastante satizfação do trabalho realizado, tendo em conta o processo e progresso do mesmo. Comançando por algoritmos mais simples e progressivamente procurando soluções em algoritmos melhores, seguindo o método empirico foi uma boa experiencia. Tendo em conta que se trabalhou com imagens, e que o dataset era reduzido, os resultados, sumariados na tabela acima, parecem fazer sentido, tendo em conta ao algoritmo a que se referem: algoritmos mais simples, não conseguem ”representar” a informação das imagens tão bem, pelo que têm uma performance pior; além disso, os algoritmos tendem a dar overfit devido principalmente ao tamanho do datset.

REFERENCES

[1] *https://www.kaggle.com/ardamavi/sign-language-digits-dataset*.
[2] *https://stackoverflow.com/a/52388406*
[3] *http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html*.
[4] *https://datascience.stackexchange.com/a/42600*
[5] *https://stats.stackexchange.com/a/159051*.
[6] *https://www.machinecurve.com*