American Sign Language Classification

Autores: André Alves (89334) e Miguel Mota (89331) Estudantes de Engenharia Informática do
Departamento de Eletrónica,Telecomunicações e Informática na Universidade de Aveiro
Tópicos de Aprendizagem Automática
Docente Pétia Georgieva

***Abstract*—O propósito deste trabalho, não é apenas construir um bom modelo de *machine learning* capaz de reconhecer qual é a letra em língua gestual, mas também descrever todo o processo e mudanças feitas para chegar aos melhores resultados. Dito isto, utilizou-se vários tipos de modelo como regressão logística e redes neurais. Para o nosso problema de classificação não-linear, conseguiu - se melhorar uma precisão de uma rede neural 80% para uma precisão de 99% com uma rede neural convolucional. Efetivamente, para chegar a esses resultados, é preciso perceber o que se passa com o nosso modelo, para isso utilizou-se *K-Fold Cross Validation* de forma a perceber a performance e desenvolvimento do nosso modelo,quais os melhores hiper parâmetros e evitar que ocorra *overfitting*. Poderá haver ainda um melhor modelo, mas existe inúmeras combinações de hiper parâmetros possíveis que torna-se computacionalmente impossível verificar todas, contudo o mais importante é compreender a evolução do nosso modelo e tentar melhorar perante as observações retiradas.**

***Index Terms*—Sign Language Recognition,Hyper Parameters Variation, Convolutional Neural Network,Feature Extracting,Gesture Recognition, Assisted Machine Learning,Computer Vision**

I. Introdução

O Kaggle [**1**] contém duas base de dados com várias imagens, sendo que uma serve para o treino do modelo e outra para teste, estas representam o alfabeto de língua gestual americana e representa um problema de multi-classe, neste caso 24 classes em que cada uma é uma letra (Fig. 1). Portanto, estas imagens , são a fonte de dados para a resolução do problema de interpretação dos gestos, na qual, para a parte de treino têm-se 27455 imagens com um tamanho de 28×28, ou seja, 784 pixeis e para a parte de testes têm-se 7172 imagens com a mesma resolução. Na figura 2, pode-se observar que os nossos dados estão aproximadamente com 1100 exemplos por cada classe, o que se pode dizer que a base de dados de treino está bem balanceada.

A interpretação de língua gestual trata-se de um problema de classificação não-linear. Desta forma, na escolha de algoritmos pode-se excluir algoritmos que sejam não-classificação e lineares. Assim, há várias opções, como por exemplo, Regressão Logística e Redes Neurais.Utilizaram-se vários algoritmos de forma a comparar o *desempenho* e *performance* de cada um,juntamente com a manipulação dos *hyper-parameters*.

Fig. 1: As 24 classes que representam cada uma uma letra do alfabeto em língua gestual.

A	B	C	D	E	F	G	H	I	J	K	L	M
1126	1010	1144	1196	957	1204	1090	1013	1162	0	1114	1241	1055
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1151	1196	1088	1279	1294	1199	1186	1161	1082	1225	1164	1118	00

Fig. 2: Quantidade de figuras por label

A	B	C	D	E	F	G	H	I	J	K	L	M
331	432	310	245	498	247	348	436	288	0	331	209	394
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
291	246	347	164	144	246	248	266	346	206	267	332	00

Fig. 3: Quantidade de figuras de teste por label

II. Regressão logística

O algoritmo de Regressão logística é aplicado quando temos problemas de classificação. Deste modo começamos por testar a eficácia deste algoritmo perante os nossos dados e rapidamente nos deparamos que o algoritmo para além de não estar a dar valores muito precisos o treino do modelo e teste eram demasiado lentos, cross-validation com apenas 1/20 das imagens demorava

cerca de 4 horas. Isto acontece porque o algoritmo não é muito adequado para problemas complexos não lineares, fazendo-nos abandonar este modelo e passamos a trabalhar com Redes Neurais.

III. Redes Neurais

Posteriormente , à tentativa de uso de *Logistic Regression* passamos a trabalhar com NN da libraria *Sklearn* este , por sua vez, sendo um modelo mais apropriado para o caso de estudo começou a demonstrar resultados mais próximos da realidade e com bastante mais rapidez. Conseguimos obter 80% com mínimo esforço usando os seguintes valores de neurônios:

$$Input \rightarrow 784 Neurônios \rightarrow 314 Neurônios \rightarrow Prediction$$

No entanto, esta livraria não nos dava, pelo que encontramos, algumas ferramentas que pretendíamos. Intencionávamos, visualizar gráficos que nos ajudariam a alterar os *hyper-parameters* tais como o *C* ou *iterations* e neste aspecto a livraria ou não possuía ou possuía métodos que era bastante lentos, isto porque obrigavam a correr,novamente, o algoritmo de outro modo para executar um objectivo em concreto. Pensamos então acelerar este processo com uso de GPU,o que nos levou a descobrir que não seria possível com **Sklearn**. Perante todos os pontos acima mencionados passamos a usar uma das livrarias mais usados de momento em *Machine Learning*, **Keras**, porque esta nos permitia mais liberdade no controlo de todo o processo.

IV. Redes Neurais Convolucionais

O algoritmo Redes Neurais Convolucionais é uma classe de rede neural do tipo *feed-forward* muito utilizado em análise de imagens digitais mais conhecido como **CNN** que vem do inglês , *Convolutional Neural Network*. Esta variante de rede neural segue a mesma estrutura, ou seja, contém uma camada de *input*, uma ou várias que são *hidden-layers* e, por fim, a camada de *output*. No entanto, as *hidden-layers* podem ter diferentes funcionalidades,no nosso caso utilizaram-se as seguintes camadas:

- ***Convolutional*** ,esta é uma camada que no caso do problema em questão, recebe como entrada uma array $28 \times \ 28 \times 1$, em que o 1 representa o número de canais, ou seja, no caso de a imagem ser RGB seria 3 canais, no entanto, neste caso como as imagens estão em *grayscale* só tem um. O objetivo de um Convolutional Layer é reduzir as imagens de uma forma a que estas sejam mais fáceis de processar sem perder features que são imprescindíveis para uma boa previsão. Nesta camada é possível definir o tamanho de um filtro,por exemplo, de dimensão 3×3, onde o filtro irá percorrer o array de pixeis, multiplicando pelos valores do filtro com os da imagem original, guardando o respetivo valor. Como resultado, produz um mapa de ativação bidimensional de filtros que são ativados quando encontrados noutra localização na imagem;
- ***Pooling*** é uma camada com objectivo de compressão, onde se pode definir as dimensões de um *pool size* que irá em cada localização da janela na imagem escolher o valor máximo. Sendo, assim, muito importante para a redução de parâmetros, memória e ajudar a evitar *overfitting*. A utilização desta camada é muito comum entre as camadas convolucionais numa arquitetura do tipo CNN.
- ***Dropout*** é uma camada de regularização , que de forma aleatória ignora alguns nós de certa camada consoante o *rate* estabelecido, que varia entre 0 e 1.
- ***Flatten*** é uma camada que transforma uma matriz bidimensional num vector para ser fornecido à próxima camada da rede neural.

Inicialmente, há um **pré-processamento dos dados** baseado numa normalização, onde se utilizou uma função da libraria *SK-learn* nomeadamente *StandartScale*, que calcula através da seguinte fórmula:

$$z = \frac{\chi - \mu}{S}$$

o χ representa as *features*, o μ a média das amostras de treino e o S a distância do mínimo valor ao máximo, na qual, neste problema é 255. Foram utilizados os dados sem ser normalizados, uma vez que a normalização tem maior eficiência quando as *features* têm *range* diferentes, contudo a diferença foi significativa, o primeiro modelo que testamos, que será explicado mais à frente, sem normalização foi de 2% , enquanto que com a normalização passou para 92%, o que é uma grande discrepância. Isto pode ser justificado porque para convergir com dados não normalizados, o algoritmo demora mais *epochs* e não converge ao mesmo tempo que os dados normalizados. Desta forma, passamos a normalizar os dados antes de treinar o modelo.

Construiu-se uma rede neural inicial de raiz, seguindo vários *standards* e com base no estudo do artigo [**2**], com a seguinte estrutura:

$$Conv2D \rightarrow MaxPooling2D \rightarrow Dropout \rightarrow Conv2D \rightarrow MaxPooling2D \rightarrow Dropout \rightarrow Flatten \rightarrow Dense \rightarrow Dropout \rightarrow Dense$$

, onde as camadas convolucionais têm uma quantidade de filtros 32 e 64, com dimensões 3×3, uma função de ativação *relu*, com uma camada procedente *Pooling* com uma dimensão 2×2 e a camada *Dropout* com uma fração de 0.25. Por fim, encontra-se a camada *Dense* de forma a regularizar a densidade de conexões da rede. Tendo isto em conta, obteve-se um resultado de precisão 93%

e de loss $0,23$ e não esquecer que estes resultados foram obtidos com 10 épocas e com um *batch size* igual a 1000. Obviamente, tendo os resultados em consideração, é um modelo "fraco" para apenas 24 classes, por esta razão passamos a variar vários hiper parâmetros para obter melhores resultados. Nesta primeira fase de execução apenas treinamos o modelo com os dados de treino e testamos com os dados de teste, não tendo assim suporte para validar o que realmente se passava com o nosso modelo, isto é, se estava a ter *high-bias* ou *high-variance*. Após esta fase inicial , começamos a fazer um processo de treino-validação mais apropriado e estruturado, usando *cross-validation* e *loss-function* para nos ajudar a compreender o que se passa com o nosso modelo.

Portanto, utilizou-se **K-Fold Cross Validation** para medir a performance do nosso modelo e através das observações feitas ,com base nos valores obtidos por este, escolher os melhores valores dos hiper parâmetros. O seu procedimento tem um parâmetro *k* que se refere as vezes que os dados são divididos em vários *folds* , se possível, do mesmo tamanho. De seguida, o modelo é treinado com *k - 1 folds*, em que o *fold* restante serve para teste. Isto é, cada grupo de dados tem oportunidade para servir para teste uma vez e usado para treino do modelo *k-1* vezes.

Começamos por fazer um treino , já a utilizar *K-Fold* com 5 *splits* , com um modelo não muito diferente do usado anteriormente,utilizamos ,portanto, camadas convolucionais com 32 filtros e *Pooling* de dimensões 2×2 e com um Dropout baixo de 0.2, tendo passado a usar 20 épocas. Com este teste obtivemos os seguintes resultados:

Test Score	0.9641
Media Validation Score	1.0
Media Train Score	0.993179752

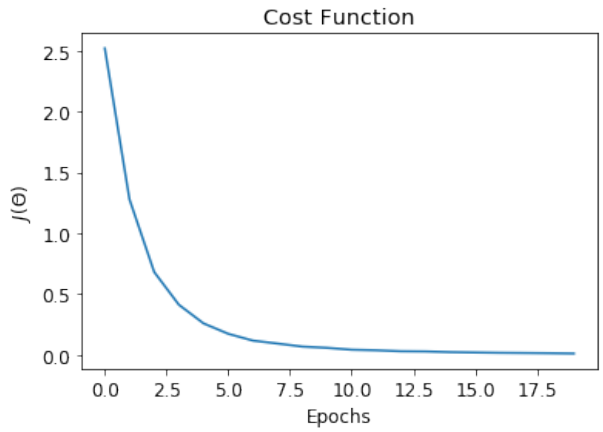

Fig. 4: Loss Function do Teste para Dropout 0.2, 32 filtros na camada convolucional e Pooling 2x2

Para uma primeira iteração não aparenta ser um resultado mau, sendo já um aumento de 3%, no entanto, tendo em conta o número de dados por label seria de esperar um resultado próximo de 100%.Podemos ver que a função convergiu de forma linear e não abrupta, por isso, o problema não se encontra no facto de o modelo não ter convergido. Tendo então em conta os valores do Cross-Validation podemos deste modo considerar que está a haver **overfit** para os dados de treino uma vez que a validação está a ser bastante boa e as previsões encontram-se ligeiramente a baixo do esperado.

Efetuamos então alterações no *hyper-parameter* dropout , que serve como regularizador, para ver as mudanças que este teria no nosso modelo, como podemos ver na figura 5 a accuracy da validação mantém-se elevada mas a do treino vai descendo quanto maior for o dropout, no entanto , criando um contraste nota-se uma elevação do valor de teste, apesar de pouco significativo, pode demonstrar que estamos a diminuir o *overfit* perante o treino feito anteriormente.

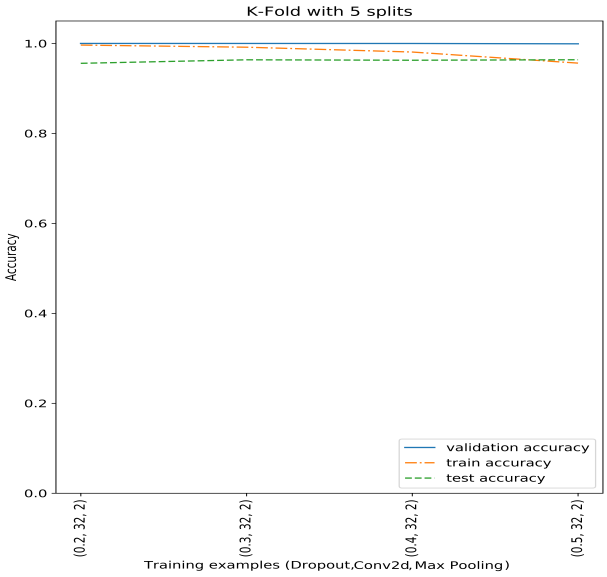

Fig. 5: Resultados obtidos alterando o dropout

Sendo os valores para 0.3, 0.4, e 0.5 bastante idênticos optamos pelo valor 0.4 uma vez que perante a figura 5 não parece estar nem a dar overfit nem a diminuir bastante a accuracy do treino. Perante esta análise passamos a usar um dropout 0.4 para as restantes alterações de *hyper-parameters*. Apesar de se notar uma pequena melhoria esta mantêm-se inferior ao que pretendíamos. Para compreender o que se passava olhamos para os valores que estavam a dar incorretos, sendo que obtivemos o seguinte (para dropout 0.4):

- 181 classificações erradas em 7172 imagens, com 11/24 classes a serem mal classificadas.

- G,T e N responsáveis por 31,45 e 30 respetivamente.
- 3/11 classes são responsáveis por cerca de 60% das classificações erradas.
- As labels mais escolhidas erradamente foram H,M e X.

Com base nos itens descritos em cima, nas imagens de cada classe e o número existente de imagens de teste por classe não variar muito , consideramos ,uma vez que o teste error se encontra mais elevado que o esperado, que o problema poderá ser por o modelo não estar a conseguir detetar algumas features importantes para a classificação. Portanto,tendo em conta isto decidimos adicionar mais filtros as camadas convolucionais de forma a que estas conseguissem detetar melhor as features e as linhas horizontais e verticais da nossa imagem. Mantendo o dropout a 0.4 repetimos o processo de *cross-validation* e *testing* com *128* e *312* filtros, confirmando que a *loss function* tinha convergido apesar de não termos gráfico que comprove. Através deste teste obtemos o seguinte gráfico:

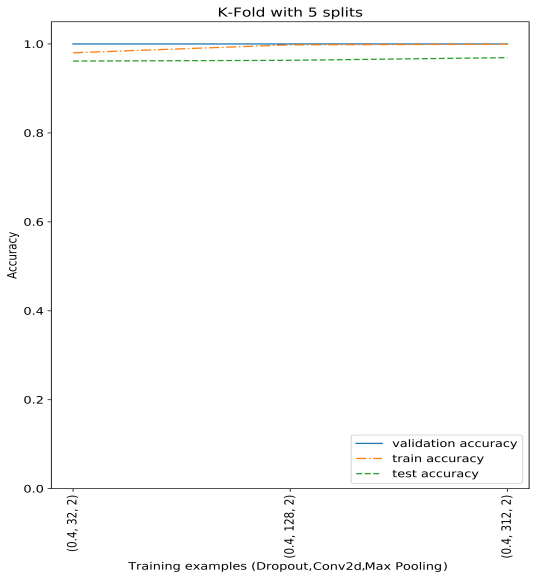

Fig. 6: Alterações feitas na camada convolucional e resultados

Conseguimos ver que o treino melhorou bastante e que o cross-validation continua alto, no entanto, para o melhor resultado de 312 filtros, enquanto o treino melhorou 4/5% o teste apenas melhorou de 0.96% para 0.97%.O valor alto de cross-validation pode ser devido ao facto de o modelo ver os dados todos pelo menos 1x.

De notar que as classes mais responsáveis pelo erro continuam a ser **G,T e N**.Através desta alteração consideramos que está a haver um pequeno *overfit* dos dados de treino. Uma vez que estamos a manter o dropout fixo a 0.4 temos que alterar o último parâmetro para tentar obter um resultado ainda mais fidedigno. Perante esta última alteração, sabemos que o output da camada convolucional aumentou substancialmente com o aumento do número de filtros de 32 para 312, desta forma com base no funcionamento do Max Pooling (Fig.7), podemos considerar que tendo um Max Pooling de 2×2 pode estar a ser o causador deste *overfitting*, pois estamos a guardar o *max* de cada 2×2 pixeis,portanto podemos experimentar aumentar o Max Pooling para 3×3 de forma a que este tenha uma função mais regularizadora. Optamos por esta alteração pois com base na nossa compreensão do paper [**4**] , da descrição detalhada de pooling e dos resultados que estávamos a obter pensamos que seria uma boa solução.

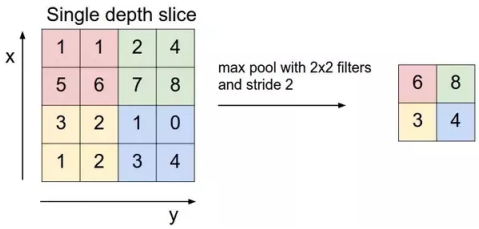

Fig. 7: Funcionamento do max pooling.

Através desta mudança conseguimos obter um resultado bastante bom , e deixamos de alterar o modelo, pois perante algumas adversidades presentes nos dados de treino e de teste,que serão referidas mais à frente, consideramos muito difícil aumentar a fiabilidade do modelo.Desta forma obtivemos os seguintes valores:

Test Score	0.9905
Media Validation Score	1.0
Media Train Score	0.9983

Devemos sublinhar que devido ao random dropout o modelo após ser *fitted* pode variar aproximadamente 0.2% na *accuracy* dos dados de treino ,pelo que executamos o algoritmo 15 vezes para obter uma média mais precisa do valor real de precisão perante os dados de teste.

Media Test Score -15 iter 0.99049079...

No final tendo em conta as alterações que fizemos, voltamos a executar o processo *K-Fold Cross-Validation* percorrendo todas as combinações dos hiper parâmetros que alteramos anteriormente entre o *rate* na camada **Dropout, número de filtros** nas camadas *Conv2D* e **tamanho do pooling** na camada MaxPooling2D para visualizar quais os valores que tornam o modelo melhor de forma a evitar acasos e confirmar que o modelo atual foi realmente o melhor. Com isto obteve-se o gráfico

8, onde se pode retirar que no início há *overfit*, pois o *text accuracy* está muito abaixo em relação ao bons resultados de *validation e train accuracy* aproximadamente 100%. Por outro lado, é observável vários picos descendentes para quando temos um número de filtros igual a 32. No entanto, obteve-se melhores resultados, ou seja, 100% de *validation* e *train accuracy* e 99% de *test accuracy*,quando se executou com um *rate* de *Dropout* igual a 0.4 (que se variou entre 0 a 0.5), isto porque é um parâmetro que ajuda a prevenir *overfit* , logo é importante para obter um resultado aproximado a 100%, número de filtros igual a 312 o que pode ser um ponto importante, pois se formos visualizar as classes, há certos pormenores que diferem entre elas, como por exemplo,se pode ver na figura 1 a classe M e N, portanto, quanto maior for a quantidade de filtros, melhor será o algoritmo a detetar esses detalhes e a diferir entre duas classes parecidas, por isso, quando se reduz esse número para 32, os valores baixam drasticamente, desta forma já não se consegue distinguir. Por fim, um *max pooling* com um *size* 3 ajuda a que o modelo com 312 filtros se torne mais *performante* e evita *overfitting* ,consequentemente, obtendo melhores modelos.

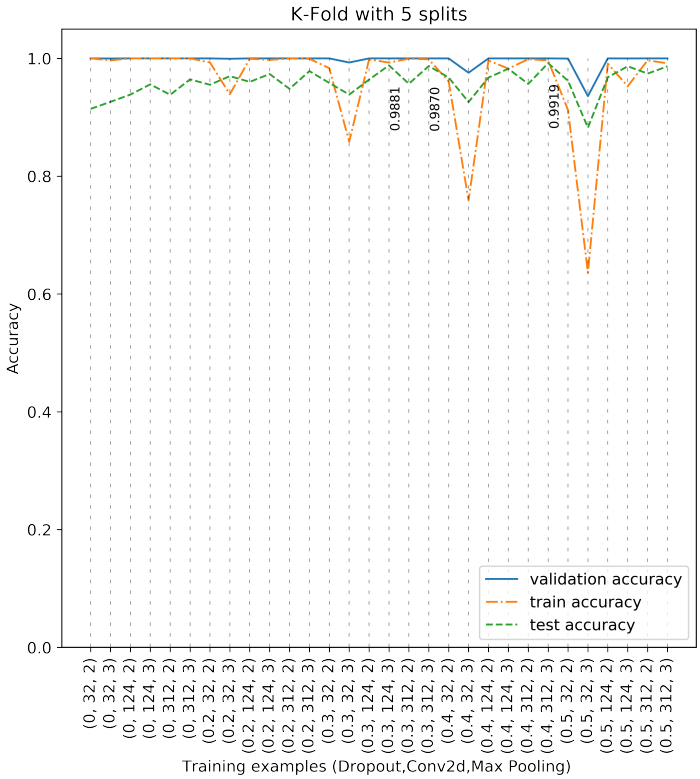

Fig. 8: Figura representativa de todas as combinações consoante as alterações que foram feitas com base nos resultados obtidos por K-Fold Cross Validation,sinalizando os 3 melhores scores.

V. Final Fine Tuning

Não possuindo recursos computacionais de larga escala, alterar ainda mais valores de hiper parâmetros, executando todas as combinações (com cross-validation), seria excessivo e difícil, desta forma apesar de não ser tão correto como a forma como testamos os parâmetros anteriores, procedemos a alterar o número de épocas procurando melhorar , minimamente, um já bom modelo. Não usamos mais épocas inicialmente porque elevam rapidamente o custo computacional e não implicam diretamente uma melhoria no modelo final. Assim, usamos os 3 melhores modelos presentes na figura 8 e iteramos 5 vezes, de forma a obtermos uma média razoável, perante vários valores de épocas procurando o melhor valor. Portanto, nas figuras 9, 10 e 11 que mostram a variação de *accuracy* perante os três melhores modelos, observa-se que em épocas diferentes têm resultados diferentes, ou seja, na figura 9 a melhor época é a 40 com um *accuracy* igual a $99,2\%$, enquanto que nos outros modelos cinquenta épocas é o melhor com uma *accuracy* igual a 99%. O melhor modelo manteve-se o mesmo,tendo melhorado 0.2%, mas podia não acontecer, poderia haver um modelo mais "fraco" em relação aos hiper parâmetros que definimos e ter obtido melhores resultados em "X" épocas. Assim finalizámos o processo, chegando ao nosso melhor modelo com hiper parâmetros de *rate* de *dropout* de 0.4, 312 filtros nas camadas *Conv2d*, e *pooling* com um *size* de 3×3 e 40 épocas atingindo uma *accuracy* de 99.2% muito próximo de cem por cento.

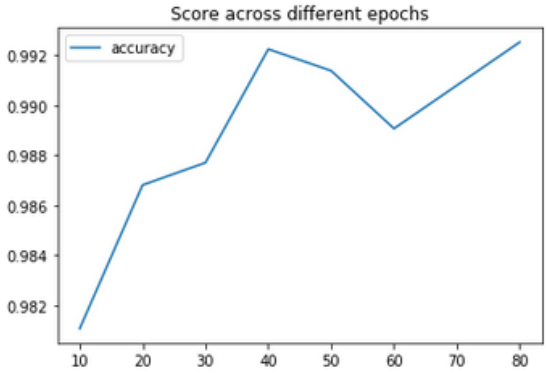

Fig. 9: Demonstra a accuracy do melhor modelo com hiper parâmetros (0.4, 312, 3) para várias épocas. Cada época correu 5 vezes para tentar obter um valor mais próximo da realidade

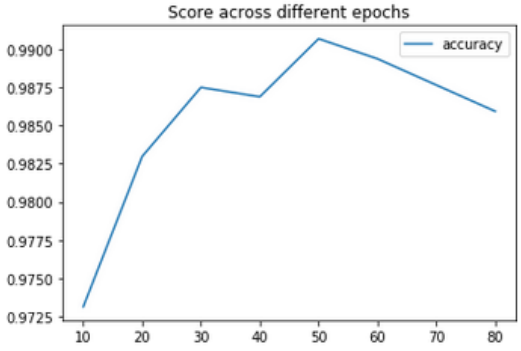

Fig. 10: Demonstra a accuracy do segundo melhor modelo com hiper parâmetros (0.3, 124, 3) para várias épocas.Cada época correu 5 vezes para tentar obter um valor mais próximo da realidade

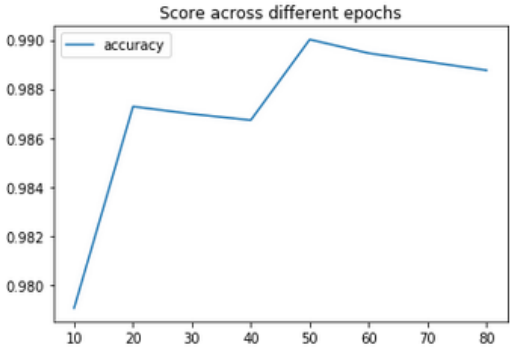

Fig. 11: Demonstra a accuracy do terceiro melhor modelo com hiper parâmetros (0.3, 312, 3) para várias épocas.Cada época correu 5 vezes para tentar obter um valor mais próximo da realidade

Perante estes últimos 3 gráficos confirmamos que o modelo ao qual tínhamos chegado foi o que apresentou as melhores previsões, perante isto procedemos a obter a *confusion-matrix*,fig. 12 das previsões feitas pelo modelo com os dados de teste. Na presença destes dados conseguimos ver que o modelo está bastante próximo da realidade, divergindo em certos valores :

- A letra G foi mal categorizada como letra Q em 0,8% dos testes
- A letra V foi mal categorizada como letra U em 5.2% dos testes
- A letra T foi mal categorizada como letra X em 8.4% dos testes
- A letra B foi mal categorizada como letra K em 0.2% dos testes
- A letra H foi mal categorizada como letra G em 0.4% dos testes

Estes variam de forma constante, não falham de forma dispersa, falhando maioritariamente para a mesma letra. Podemos também com base no que foi dito anteriormente ver que *a trend* da letra T (juntamente com a leta V) ser das mais responsáveis por falhas se mantêm. O mesmo se pode dizer para as classes que são escolhidas mais vezes erradamente mantendo-se no *top* a letra X. A figura 12 totaliza uma accuracy de 99.37%.

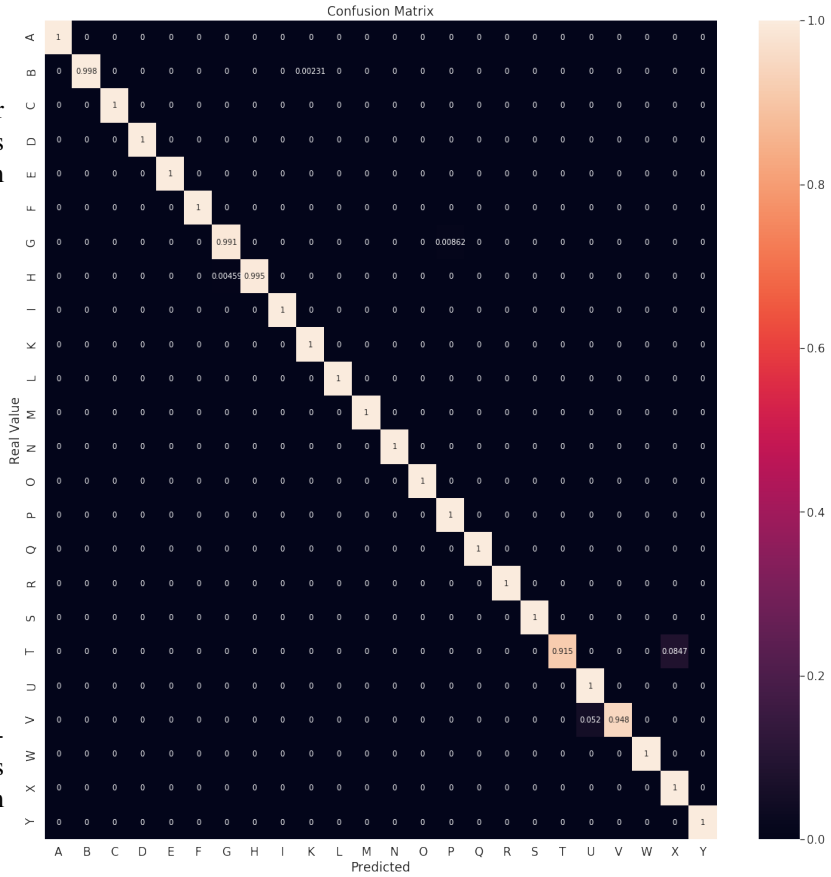

Fig. 12: Confusion-Matrix para o modelo com os hiper parâmetros (0.4,312,3) e 40 épocas de treino

VI. Conclusion

Através do processo efetuado para chegar a um bom modelo conseguimos entender quando e que parâmetros mudar em cada situação , de forma a obter o modelo mais próximo da realidade. Aprendemos como usar métodos estatísticos e a importância de o fazer. Neste caso em concreto *cross-validation* para treinar um modelo antes de efetuar uma previsão para o modelo de treino, apesar deste não ter sido tao influente no nosso modelo como esperávamos. Perante o estudo feito podemos verificar que conseguimos obter um modelo bastante bom para grande parte das situações, no entanto, consideramos que o modelo podia ainda ser mais preciso se não houvessem algumas falhas no modelo de dados. Por

exemplo na figura da letra 13 que representa alguns modelos de K, que acaba por ser uma das *labels* que se confunde com B, consegue-se visualizar que alguns modelos são difíceis de identificar mesmo comparando com a 1,por exemplo **e.g coluna 1, linha 3**. Dando atenção agora a imagem das classes H, na figura da 14, podemos ver que algumas delas não se distingue a existência de 1 ou 2 dedos,**e.g coluna 1, linha 4** podendo inferir o modelo em erro. Agora tendo em conta a imagem das classes Y, na figura 15, podemos ver que há classes que estão completamente erradas , **coluna 2, linha 3**, neste caso não afetou a *performance*, no entanto havendo erros em 3 classes podem existir outros que não detetamos e que podem afetar os resultados demonstrados anteriormente. No entanto, mesmo com estas dificuldades os erros cometidos, representados na figura 16, não seriam esperados por isso talvez fosse necessário mais dados para estas classes.

Fig. 13: Modelo de dados de treino para a label K

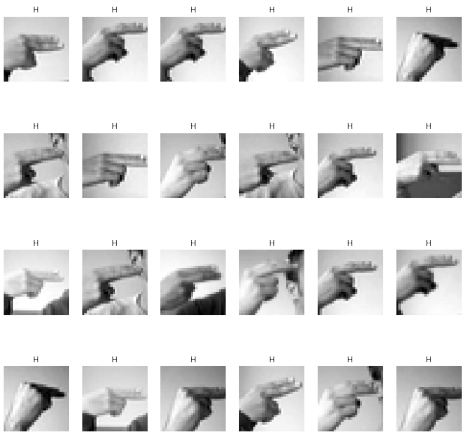

Fig. 14: Modelo de dados de treino para a label H

Fig. 15: Modelo de dados de treino para a label Y

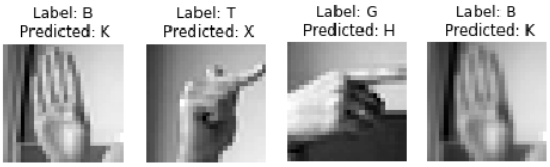

Fig. 16: Erros no modelo

VII. Novelty and contributions

Perante os *papers* analizados , apesar de nem todos estarem a usar a mesma base de dados (visto que não encontramos papers suficientes usando a nossa base de dados [**1**]), não efetuamos nenhuma *novelty* ou *contributions* no nosso trabalho, pois mantivemo-nos bastante próximos do que tem vindo a ser realizado no contexto de classificação de imagens, contudo perante o que vai ser descrito embaixo consideramos ter possuído algum impacto positivo a nível de *accuracy*.

Comparativamente aos *papers* com *dataset* diferente: no *paper* [**5**] o grupo de investigadores afirma obter 82% de *precision*; no paper [**6**] o grupo de investigadores possui uma CNN semelhante à nossa, mas que apenas obteve 82.5% de accuracy. Tendo em consideração que nós obtivemos *recall* e *accuracy* a rondar os 99% , caso estivéssemos a falar do mesmo *dataset* diríamos que o nosso estaria superior, não tendo referências que estávamos a usar o mesmo *dataset* que os investigadores destes papers , apenas podemos utilizar isto como uma mera comparação pouco fidedigna.

Passando agora a comparar com o paper [**3**] que consideramos que usou o mesmo dataset (este encontra-se nas referencias do próprio), com o nosso modelo conseguimos uma accuracy média de 99.2%, enquanto no paper o melhor valor apresentado foi com uma KNN ,obtendo um valor de 98.03%. Desta forma podemos concluir que para o mesmo dataset conseguimos ter

alguma contribuição de valor ,pois através do nosso método de treino e alterações efetuadas , com base nos gráficos mencionados anteriormente , conseguimos obter um modelo 1.2% mais eficaz.

Existem também alguns repositórios online e websites que possuem métodos de classificação deste modelo de dados, no entanto estes encontravam-se maioritariamente a rondar os 88%/90%, havendo um que tinha 95% nos dados de teste. Tendo isto e a nossa *accuracy* em conta podemos considerar que o nosso modelo e metodologia de treino foi superior.

Concluindo, podemos considerar que com o nosso trabalho contribuímos minimamente em termos de performance e optimização do modelo de dados , MNIST American Sign Language [**1**],obtendo valores superiores aos existentes na internet e em papers. Apesar de termos uma accuracy ,na nossa opinião bastante boa, há sempre pontos que poderiam ser melhorados.Neste caso consideramos que a nossa metodologia juntamente com informação extra proveniente de um *kinect* (idea retirada da leitura do paper [**7**]),conseguíssemos obter valores ainda melhores uma vez que este ajuda bastante na detecção das linhas e articulações das mãos(figura 3.10,pagina 26, *paper* [**7**]) sendo estas *features* essenciais para a classificação.

REFERENCES

[1] tecperson, Sign Language MNIST, Drop-In Replacement for MNIST for Hand Gesture Recognition Tasks, https://www.kaggle.com/datamunge/sign-language-mnist/

[2] Rawini Dias, LaShay Fontenot, Katie Grant, Chris Henson, and Shivank Sood. American Sign Language Hand Gesture Recognition (2019), https://towardsdatascience.com/american-sign-language-hand-gesture-recognition-f1c4468fb177

[3] Umair Muneer Butt,Basharat Husnain,Usman Ahmed,Arslan Tariq,Iqra Tariq,Muhammad Aadil Butt, Dr. Muhammad Sultan Ziam. Feature based Algorithmic Analysis on American Sign Language Dataset (2019)

[4] Anne Bonner, The Complete Beginner's Guide to Deep Learning: Convolutional Neural Networks and Image Classification (2019) , https://towardsdatascience.com/wtf-is-image-classification-8e78a8235acb

[5] Sunil Vadera ,Salem Ameen. A Convolutional Neural Network to Classify American Sign Language Fingerspelling from Depth and Colour Images(2017)

[6] Vivek Bheda, N. Dianna Radpour. Using Deep Convolutional Networks for Gesture Recognition in American Sign Language

[7] Cao Dong,American Sign Language Language Alphabet Recognition using Microsoft Kinect(2015), https://scholarsmine.mst.edu/cgi/viewcontent.cgi?article=8391context=masters_theses