

UNIVERSIDADE DE AVEIRO

Compreender a Língua Gestual Americana

Renato Valente, Rita Amante

Mestrado Integrado em Engenharia de Computadores e Telemática

Aprendizagem Automática (40430)



Departamento de Eletrónica, Telecomunicações e Informática

2020-2021

O presente relatório foi desenvolvido no âmbito da unidade curricular, do 4º ano, do Mestrado Integrado em Engenharia de Computadores e Telemática (MIECT), Aprendizagem Automática .

Alunos

Renato Valente, Nº 89077, renatovalente5@ua.pt

Rita Amante, Nº 89264, rita.amante@ua.pt

Professora

Petia Georgieva, petia@ua.pt



Departamento de Eletrónica, Telecomunicações e Informática

2020-2021

Conteúdo

Resumo	1
Palavras-Chave	2
Lista de Figuras	3
1 Introdução	4
2 Descrição do problema	5
2.1 Motivação para o desenvolvimento do projeto	5
2.2 Problema de regressão ou classificação?	5
2.3 Descrição do conjunto de dados	6
2.4 Pré-processamento dos dados	7
3 Algoritmo de Neural Network (NN)	8
3.1 Funcionamento do algoritmo NN	8
3.2 Processamento dos dados no modelo CNN	9
3.3 Construção do modelo CNN	10
3.4 Treino do modelo CNN	11
3.5 Desempenho do modelo CNN	11
3.5.1 Modelo CNN 1	11
3.5.2 Modelo CNN 2	12
3.5.3 Modelo CNN 3	13
3.5.4 Modelo CNN 4	13
3.5.5 Modelo CNN 5	15
4 Logistic Regression (LR)	16
4.1 Funcionamento do algoritmo LR	16
4.2 Processamento dos dados no modelo LR	17
4.3 Construção do modelo LR	19
4.4 Treino do modelo LR	21
4.5 Previsão do modelo LR	22
4.6 Desempenho do modelo LR	23
4.6.1 Modelo LR 1	23
4.6.2 Modelo LR 2	23
4.6.3 Modelo LR 3	24
4.6.4 Modelo LR 4	24
5 Análise dos Resultados	25
6 Conclusões	27
Contribuição dos alunos	28
Referências	29

Resumo

Neste projeto vamos explorar o tema "American sign language understanding" com Machine Learning (ML) para tornar possível a previsão de imagens da língua gestual com a utilização de Neural Network (NN) e Logistic Regression (LR). Serão explorados estes 2 modelos de classificação de dados para obtenção do modelo mais preciso na previsão destas imagens. Para esse efeito serão variados e ajustados os hiperparâmetros de cada um destes dois modelos.

Palavras-Chave

MIECT Mestrado Integrado em Engenharia de Computadores e Telemática

ML Machine Learning

MSE Mean Squared Error

ASL American Sign Language

MNIST Modified National Institute of Standards and Technology

CSV Comma-Separated Values

NN Neural Network

CNN Convolutional Neural Network

ReLU Rectified Linear Unit

LR Logistic Regression

SGD Stochastic Gradient Descent

Lista de Figuras

1	Alfabeto representado na língua gestual americana.	5
2	Língua gestual americana.	6
3	Conjunto inicial de dados de treino.	7
4	Matrizes dos rótulos dos dados de treino e de teste.	7
5	Rede neural simples.	8
6	Modelo 1 (CNN) - Evolução da perda e Precisão.	11
7	Modelo 1 (CNN) - Normalized Confusion matrix.	12
8	Modelo 1 (CNN) - Non-Normalized Confusion matrix.	13
9	Modelo 1 (CNN) - Previsão da 25 imagens.	14
10	Modelo 2 (CNN) - Evolução da perda e Precisão.	14
11	Modelo 2 (CNN) - Normalized Confusion matrix.	15
12	Modelo 2 (CNN) - Non-Normalized Confusion matrix.	16
13	Modelo 2 (CNN) - Previsão da 25 imagens.	17
14	Modelo 4 (CNN) - Evolução da perda e Precisão.	17
15	Modelo 4 (CNN) - Normalized Confusion matrix.	18
16	Modelo 4 (CNN) - Non-Normalized Confusion matrix.	19
17	Modelo 4 (CNN) - Previsão da 25 imagens.	20
18	Modelo 5 (CNN) - Evolução da perda e Precisão.	20
19	Modelo 5 (CNN) - Normalized Confusion matrix.	21
20	Modelo 5 (CNN) - Non-Normalized Confusion matrix.	22
21	Modelo 5 (CNN) - Previsão da 25 imagens.	23
22	Modelo 1 (LR) - evolução de perda e de precisão.	24
23	Modelo 2 (LR) - evolução de perda e de precisão.	24
24	Modelo 3 (LR) - evolução de perda e de precisão.	25
25	Modelo 4 (LR) - evolução de perda e de precisão.	25

1 Introdução

A língua gestual é uma linguagem visual completa e organizada que se expressa através dos movimentos das mãos, maioritariamente. Este meio de comunicação é principalmente utilizado por pessoas surdas e/ou mudas para comunicarem entre si. Na verdade, este método começa a ser utilizado para outros fins, como por exemplo, na aprendizagem de crianças com problemas na fala. No entanto, compreender esta linguagem não é uma habilidade universal.

Assim, surge este projeto que pretende construir um sistema que reconheça o alfabeto americano, através dos gestos das mãos, de forma a facilitar a comunicação entre qualquer indivíduo.

O presente relatório pretende compreender a língua gestual americana através de ML, desenvolvida no âmbito da unidade curricular de Aprendizagem Automática. No seu decorrer serão clarificados alguns aspectos importantes, assim como: motivação para o desenvolvimento do projeto, descrição teórica e prática dos modelos implementados, entre outros.

Para a realização deste projeto foram utilizadas as seguintes ferramentas de trabalho: Anaconda3 para Python3 e Jupyter Notebook. Foi também preciso instalar algumas bibliotecas, executando os seguintes comandos:

- pip install tensorflow
- pip install keras
- pip install opencv-python
- pip install torch
- pip install torchvision
- pip install jovian

2 Descrição do problema

2.1 Motivação para o desenvolvimento do projeto

A língua gestual é uma linguagem visual completa e organizada, onde a forma, o posicionamento e o movimento das mãos desempenham papéis importantes para a comunicação entre qualquer indivíduo. No entanto, nem todas as pessoas compreendem esta linguagem.

A motivação subjacente à escolha deste tema, centra-se na dificuldade existente na comunicação entre as pessoas que compreendem e utilizam a língua gestual e as pessoas que comunicam apenas em língua oral, impedindo que os gestos sejam corretamente compreendidos.

O objetivo deste projeto assenta na contribuição para colmatar a barreira de comunicação existente entre pessoas que comunicam utilizando a Língua Gestual Americana e pessoas que comunicam utilizando a língua oral. Assim sendo, pretende-se construir um sistema que reconheça o alfabeto americano (figura 1), através dos gestos das mãos, de forma a facilitar a comunicação entre qualquer indivíduo.

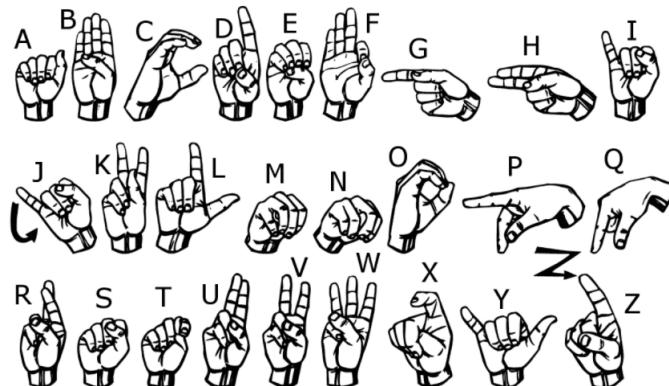


Figura 1: Alfabeto representado na língua gestual americana.

Fonte: Sign Language MNIST

2.2 Problema de regressão ou classificação?

O algoritmo de ML que se vai utilizar irá aprender sobre os dados para resolver um problema. Esse problema poderá ser de dois tipos: regressão ou classificação.

Um **problema de regressão** existe quando é preciso prever um valor numérico específico, onde o modelo poderá apresentar como resposta qualquer valor.

Portanto, a modelação de regressão é a tarefa de aproximar uma função de mapeamento (f) das variáveis de entrada (X) para uma variável de saída contínua (Y). Uma variável de saída contínua é um valor real, como um número inteiro ou valor de vírgula flutuante.

Como este problema prevê uma quantidade, a habilidade do modelo deve ser relatada como um erro nessas previsões, calculando, por exemplo, o Mean Squared Error (MSE).

Por outro lado, um **problema de classificação** existe quando é preciso encontrar uma classe, dentro das possibilidades limitadas existentes. Na verdade, pode-se prever valores numéricos nos problemas de classificação, mas nestes casos, esta previsão significará sempre uma categoria, ou seja, este número poderia ser substituído por qualquer letra, palavra ou mesmo outro número, sem prejudicar a compreensão das previsões.

Portanto, a modelação de classificação é a tarefa de aproximar uma função de mapeamento (f) das variáveis de entrada (X) para variáveis de saída discretas (Y). As variáveis de saída geralmente são chamadas de rótulos ou categorias. A função de mapeamento prevê a classe ou categoria para uma determinada observação.

Visto isto, o problema deste projeto é um problema de classificação e para a sua implementação recorreu-se a dois algoritmos de **classificação**: algoritmo Convolutional Neural Network (CNN) e LR.

2.3 Descrição do conjunto de dados

Neste projeto, usou-se o conjunto de dados American Sign Language (ASL) fornecido pelo Modified National Institute of Standards and Technology (MNIST), disponível publicamente no Kaggle. A base de dados MNIST de língua gestual americana é utilizada para preparar diferentes sistemas de tratamento de imagens e para preparação e testes no campo de ML.

O conjunto de dados no Kaggle está disponível no formato Comma-Separated Values (CSV), onde os dados de treino têm 27455 linhas e 785 colunas e os dados de teste têm 7172 linhas e 785 colunas. A primeira coluna do conjunto de dados representa o rótulo da classe da imagem e as 784 colunas restantes representam uma imagem de 28x28 pixels, com valores em tons de cinza entre 0-255 (pixel1, pixel2 ... pixel784).

Essas imagens pertencem às 24 classes do alfabeto americano, começando de A a Y (figura 2), sem os rótulos das classes 9 (correspondente à letra J) e 25 (correspondente à letra Z) por causa dos movimentos dos gestos.



Figura 2: Língua gestual americana.

Fonte: Sign Language MNIST

2.4 Pré-processamento dos dados

A fim de obter o conjunto de dados, foram lidos 27455 dados de treino do ficheiro *sign_mnist_train.csv* para a variável *dataTrain* (figura 3) e 7172 dados de testes do ficheiro *sign_mnist_test.csv* para a variável *dataTest*.

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782	pixel783	pixel784
0	3	107	118	127	134	139	143	146	150	153	...	207	207	207	207	206	206	206	204	203	202
1	6	155	157	156	156	156	157	156	158	158	...	69	149	128	87	94	163	175	103	135	149
2	2	187	188	188	187	187	186	187	188	187	...	202	201	200	199	198	199	198	195	194	195
3	2	211	211	212	212	211	210	211	210	210	...	235	234	233	231	230	226	225	222	229	163
4	13	164	167	170	172	176	179	180	184	185	...	92	105	105	108	133	163	157	163	164	179

5 rows × 785 columns

Figura 3: Conjunto inicial de dados de treino.

Após a obtenção das matrizes dos rótulos, removeu-se das variáveis *dataTrain* e *dataTest* os seus rótulos, guardando as matrizes resultantes (apenas com os dados dos 784 pixeis) nas variáveis *xTrain* e *xTest*.

A coluna 'label' no conjunto de dados fornece as informações sobre o alfabeto que a imagem representa. Assim, guardou-se em duas matrizes, os rótulos correspondentes ao conjunto de dados de treino (*yTrain*) e de teste (*yTest*).

No modelo CNN foi usada a biblioteca *sklearn* para conversão das matrizes em matrizes binárias, função *LabelBinarizer()* e *fit_transform*, para a futura construção da CNN. No modelo LR utilizou-se a biblioteca *tourch* para a manipulação das matrizes e futura inserção no modelo.

Em ambos os modelos o conjunto de dados de treino foi dividido num subconjunto de treino e validação, 15% do conjunto de dados de treino original foi colocado no subconjunto de validação. Ficamos com um total de 23337 dados para treino e 4118 dados para validação.

Na figura 4 ilustrada abaixo, é apresentado um gráfico que corresponde à quantidade de dados de treino para cada classe.

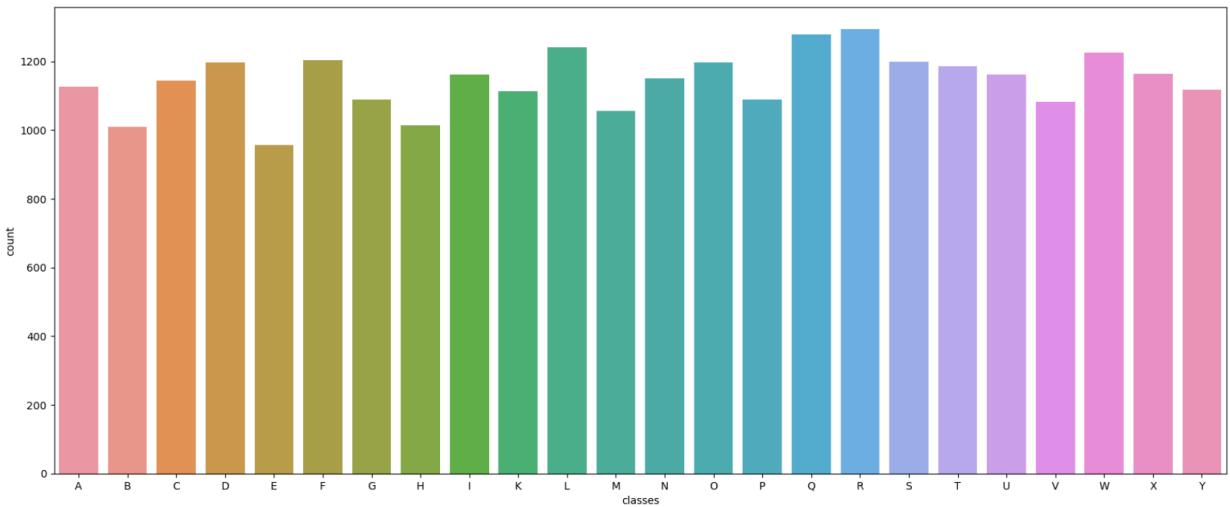


Figura 4: Matrizes dos rótulos dos dados de treino e de teste.

3 Algoritmo de Neural Network (NN)

Uma **NN** é um sistema de computação com nós interconectados que funcionam como os neurónios do cérebro humano. Através de algoritmos, estas redes podem reconhecer padrões e correlações em dados brutos, agrupá-los e classificá-los.

3.1 Funcionamento do algoritmo NN

Uma rede neural simples inclui uma camada de entrada, outra de saída e, entre elas, uma camada oculta, como ilustrado na figura 5. Estas camadas são conectadas através de nós e essas conexões formam uma rede de nós interconectados, ou seja, uma rede neural.

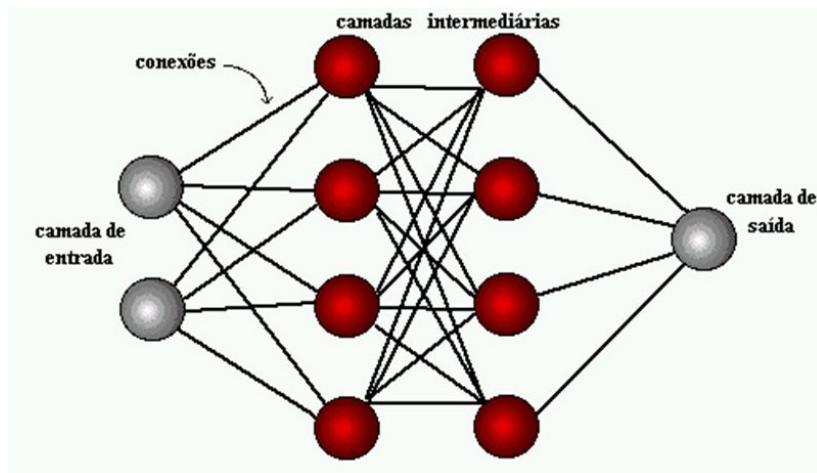


Figura 5: Rede neural simples.

Os nós são ativados quando há estímulos ou entradas suficientes. Essa ativação espalha-se através da rede, criando uma resposta ao estímulo (resultado). As conexões entre esses neurónios artificiais agem como sinapses simples, fazendo os sinais serem transmitidos de um para o outro. Há sinalização entre camadas conforme eles viajam da primeira (de entrada) até a última (de resultado) – e são processados ao longo do caminho.

Os neurónios lêem todos os dados e decidem onde estão as relações mais fortes. No tipo mais simples de uma rede, as entradas de dados recebidas são somadas e, se a soma for maior que um certo valor, o neurónio “dispara” e ativa os neurónios conectados a ele.

Com base nas camadas ocultas dentro de uma rede neural, o computador pode ser treinado para emular tarefas humanas com precisão, como reconhecer fala, identificar imagens ou realizar previsões. Igualmente, o computador pode aprender por si só a reconhecer padrões em muitas camadas de processamento, através de ML.

Resumindo, os dados são inseridos numa rede neural através da camada de entrada, que comunica com as camadas ocultas. O processamento acontece nessas camadas através de um sistema de conexões ponderadas. Os nós nas camadas ocultas combinam os dados da camada de entrada com um conjunto de coeficientes e atribui diferentes pesos para as entradas. Os

resultados dessas entradas avaliadas são, então, somados. A soma passa pela função de ativação de um nó, que determina a extensão em que um sinal deve progredir na rede para afetar o resultado final. Finalmente, as camadas ocultas ligam-se a camada de saída, de onde os resultados são obtidos.

Uma das técnicas usadas neste projeto foi a técnica de CNN. Esta rede aprende sobre pequenos recursos da imagem. Por exemplo, se um rosto humano for treinado na CNN, ela aprenderá as pequenas características do rosto, como linhas, na fase inicial da rede. À medida que sobe na hierarquia, começa a aprender características mais complexas como olhos, nariz, lábios, entre outros. E, na camada final, prevê se é um rosto ou não.

Uma CNN tem parâmetros de aprendizagem semelhantes à rede neural simples:

- **Camada de entrada:** esta camada contém uma entrada de imagem de forma (altura, largura, canais), onde a primeira e a segunda dimensões são a altura e a largura da imagem de entrada e a terceira dimensão é para RGB.
- **Camada de convolução:** esta camada calcula o volume de saída computando o produto escalar entre todos os filtros e o patch da imagem.
- **Camada de ativação:** esta camada é uma função de ativação que decide o valor final do neurônio. Depois de aplicar a convolução, pode ser possível que o produto escalar produza valores negativos. Para remover esses valores negativos, muitos CNNs usam a ativação ReLu. Isso torna todos os valores negativos iguais a zero. O volume de saída permanece o mesmo.
- **Camada Pooling:** esta camada tem como função reduzir progressivamente o tamanho espacial da representação para reduzir o número de parâmetros e cálculos na rede. Opera em cada mapa de feições independentemente. A abordagem mais comum usada em *pooling* é o pooling máximo, que agrupa o valor máximo do patch do tamanho do filtro fornecido, e outra é o pooling médio, que agrupa a média de todos os valores do patch do tamanho do filtro.
- **Camada totalmente conectada:** esta camada é uma camada neural simples que recebe entrada e produz o volume correspondente ao número de classes.
- **Camada de exclusão:** esta camada é usada para reduzir o sobreajuste no treino.

3.2 Processamento dos dados no modelo CNN

Primeiramente, foram lidos os conjuntos de dados de treino e teste, como referido no subcapítulo 2.4.

Estes conjuntos de dados, *xTrain* e *xTest* foram redimensionados para se obter matrizes de 28x28.

Os rótulos das matrizes foram também convertidos para binários, onde '1' significa que pertence à classe e '0' significa que não pertence à classe. Para isto utilizou-se a função *Label-Binarizer* e *fit_transform* da biblioteca *sklearn*.

Do conjunto de dados de treino original correspondente a 27455 dados, foram retirados 15% para dados de validação do modelo, ficamos assim com um total de 23337 dados para treino e 4118 dados para validação.

3.3 Construção do modelo CNN

Neste projeto, classificou-se os símbolos da língua gestual por meio da CNN. Após o treino bem-sucedido do modelo CNN, o alfabeto correspondente de um símbolo de língua gestual será previsto. Avaliou-se o desempenho de classificação de nosso modelo usando matrizes de confusão normalizadas e não normalizadas. Por fim, obteve-se a pontuação de precisão da classificação do modelo CNN.

A partir da definição da documentação *tensorflow.keras*, o modelo *Sequencial* é uma stack linear de camadas. O parâmetro *kernel_size* é um inteiro que especifica o mesmo valor para todas as dimensões espaciais, ou tuplo/lista de 2 inteiros, que especifica a altura e largura da janela de convolução 2D. O parâmetro *activation* é uma função de ativação a ser aplicada. O parâmetro *input_shape* é um tensor com a forma: *batch_shape* + (linhas, colunas, canais), onde é preciso usar as dimensões da imagem de entrada.

A camada de entrada do modelo obterá imagens de tamanho (28,28,1) onde '28,28' são a altura e a largura da imagem, respectivamente, enquanto '1' representa o canal de cor da imagem para tons de cinza. A camada de saída do modelo terá 24 neurónios para 24 letras diferentes, e a função de ativação será *softmax*, pois é um problema de classificação multiclasse.

A primeira camada oculta é composta por vários nós, cada um dos quais tem uma soma ponderada dos 784 valores de entrada. A soma ponderada das entradas é inserida numa função de ativação Rectified Linear Unit (ReLU), que produzirá '0' quando a entrada for negativa, mas não mudará a entrada de outra forma. As saídas do ReLU servirão como entradas para a próxima camada oculta na rede.

À medida que os dados continuam a percorrer as camadas ocultas, a rede neural tenta extrair mais recursos abstratos.

Depois que os dados passam pelas camadas Convolution (*Conv2D*) e MaxPool (*MaxPooling2D*) da rede neural, eles entram nas camadas *Flatten* e *Dense*. Essas camadas são responsáveis por reduzir os dados a uma dimensão e identificar a classe de uma imagem.

O *Dropout* foi definido como 0,2 no final. No backend, este parâmetro serve para evitar um modelos *underfit* ou problemas como o *overfitting*, ou seja, o fenómeno quando o modelo tem um desempenho muito bom nos dados de treino, mas falha miseravelmente com os dados de teste.

Uma época é uma única passagem por todos os dados de treino. Na primeira época, a rede neural estima um valor para cada peso. Para cada época subsequente, a rede neural atualiza esses pesos com valores que reduzem a perda geral.

3.4 Treino do modelo CNN

Depois que a arquitetura da CNN foi definida, procurou-se entender que fatores que afetam a precisão do modelo *CNN*. Sendo assim, criaram-se cinco modelos para verificar se o número de *epochs* e se o valor de *dropout* afetam a precisão o modelo.

Um *epoch* ocorre quando todo o conjunto de dados de treino é passado através da rede neural uma vez. A percentagem de *dropout* refere-se a quantidade de unidades, de um certo conjunto de neurónios, na rede neuronal que vão ser ignoradas durante a fase de treino, escolhidos aleatoriamente. Como temos 23337 dados de treino, e um *batch size* de 128, o número de iterações por 1 *epoch* será de 183 iterações.

O modelo 1 foi treinado para 20 *epochs* (ou seja, 20 iterações) e com *dropout* de 0.2, o modelo 2 foi treinado para 50 *epochs* e com *dropout* de 0.2 e o modelo 3 foi treinado para 100 *epochs* e com *dropout* de 0.2. Estes três modelos, serviram para verificar qual o impacto do número de *epochs* no modelo CNN.

O modelo 4 foi treinado para 50 *epochs* e com *dropout* de 0 e o modelo 5 foi treinado para 50 *epochs* e com *dropout* de 0.5. Estes dois modelos, juntamente com o modelo 2, serviram para verificar qual o impacto do *dropout* no modelo CNN.

Depois de cada modelo CNN pronto, foi preciso treiná-los alimentando o conjunto de treino.

É de salientar que o otimizador utilizado foi o *adam*.

3.5 Desempenho do modelo CNN

3.5.1 Modelo CNN 1

Este modelo foi treinado para 20 *epochs* e com *dropout* de 0.20.

De seguida, avaliou-se o modelo usando o conjunto de dados de teste e obteve-se um valor de precisão de 0.942.

Depois do treino acabar, desenhou-se a variação de precisão com a época (figura 6).

De seguida, na figura 7 é apresentada a matriz de confusão normalizada e na figura 8 é apresentada a matriz de confusão não-normalizada. Estas matrizes servem para entender melhor os pontos fortes e fracos desse modelo.

A figura 9 corresponde à visualização das previsões dos rótulos das classes para algumas imagens de teste deste modelo.

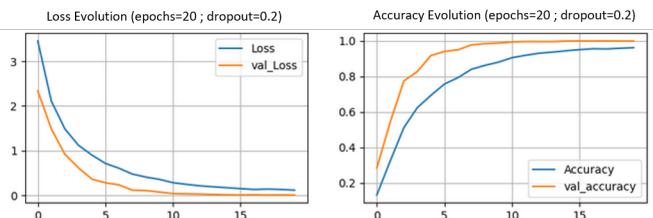


Figura 6: Modelo 1 (CNN) - Evolução da perda e Precisão.

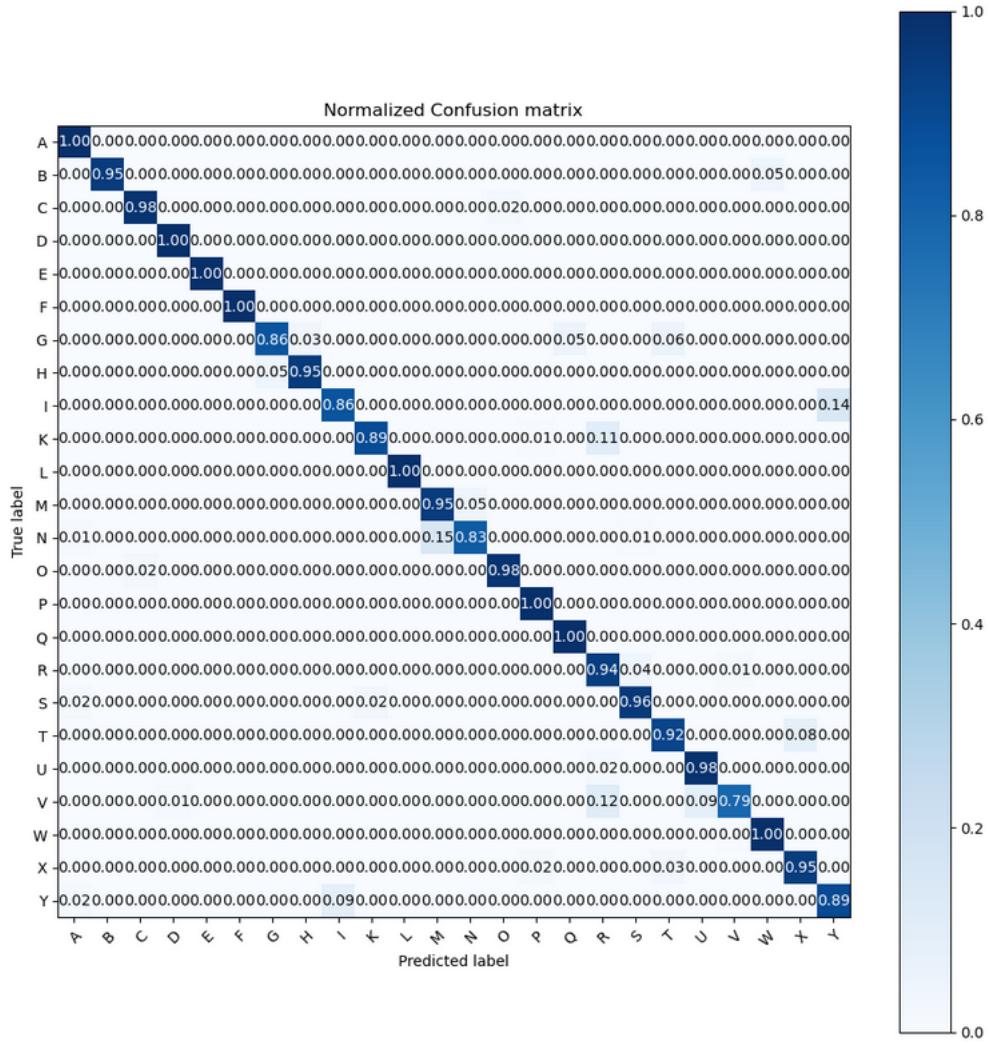


Figura 7: Modelo 1 (CNN) - Normalized Confusion matrix.

3.5.2 Modelo CNN 2

Este modelo foi treinado para 50 epochs e com dropout de 0.20.

De seguida, avaliou-se o modelo usando o conjunto de dados de teste e obteve-se um valor de precisão de 0.954.

Depois do treino acabar, desenhou-se a variação de precisão com a época (figura 10).

De seguida, na figura 11 é apresentada a matriz de confusão normalizada e na figura 12 é apresentada a matriz de confusão não-normalizada.

A figura 13 corresponde à visualização das previsões dos rótulos das classes para algumas imagens de teste deste modelo.

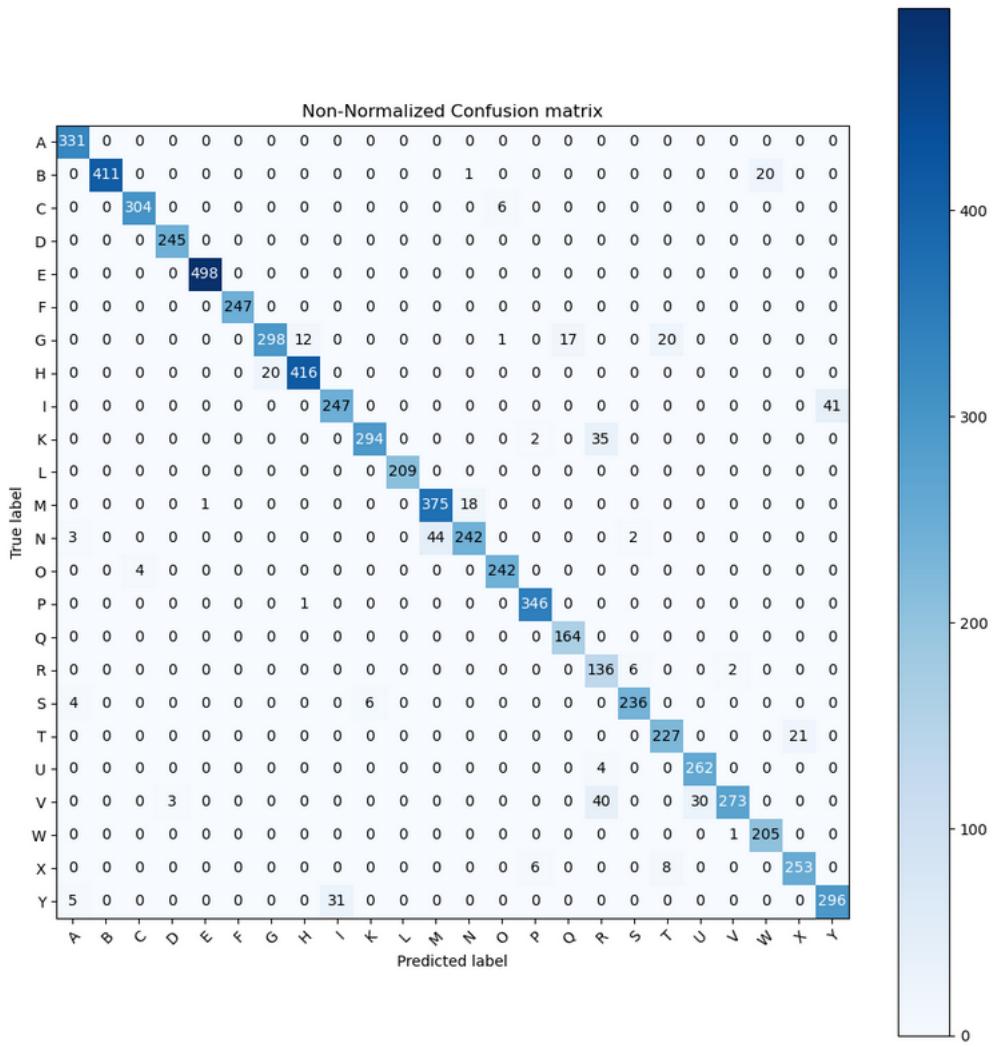


Figura 8: Modelo 1 (CNN) - Non-Normalized Confusion matrix.

3.5.3 Modelo CNN 3

Este modelo foi treinado para 100 epochs e com dropout de 0.20.

De seguida, avaliou-se o modelo usando o conjunto de dados de teste e obteve-se um valor de precisão de 0.959.

Os modelos de variação de precisão são muito semelhantes aos do modelo 2, tal como a matriz de confusão normalizada e a matriz de confusão não-normalizada. A partir de 20 *epochs* podemos concluir que o valor de previsão tende a estabilizar em 0.95.

3.5.4 Modelo CNN 4

Este modelo foi treinado para 50 epochs e com dropout de 0.

De seguida, avaliou-se o modelo usando o conjunto de dados de teste e obteve-se um valor



Figura 9: Modelo 1 (CNN) - Previsão da 25 imagens.

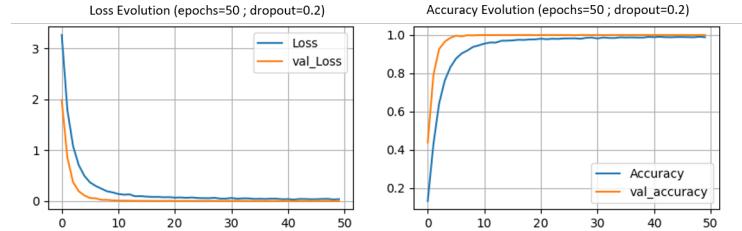


Figura 10: Modelo 2 (CNN) - Evolução da perda e Precisão.

de precisão de 0.893.

Depois do treino acabar, desenhou-se a variação de precisão com a época (figura 14).

De seguida, na figura 15 é apresentada a matriz de confusão normalizada e na figura 16 é apresentada a matriz de confusão não-normalizada. Estas matrizes servem para entender melhor os pontos fortes e fracos desse modelo.

A figura 17 corresponde à visualização das previsões dos rótulos das classes para algumas imagens de teste deste modelo.

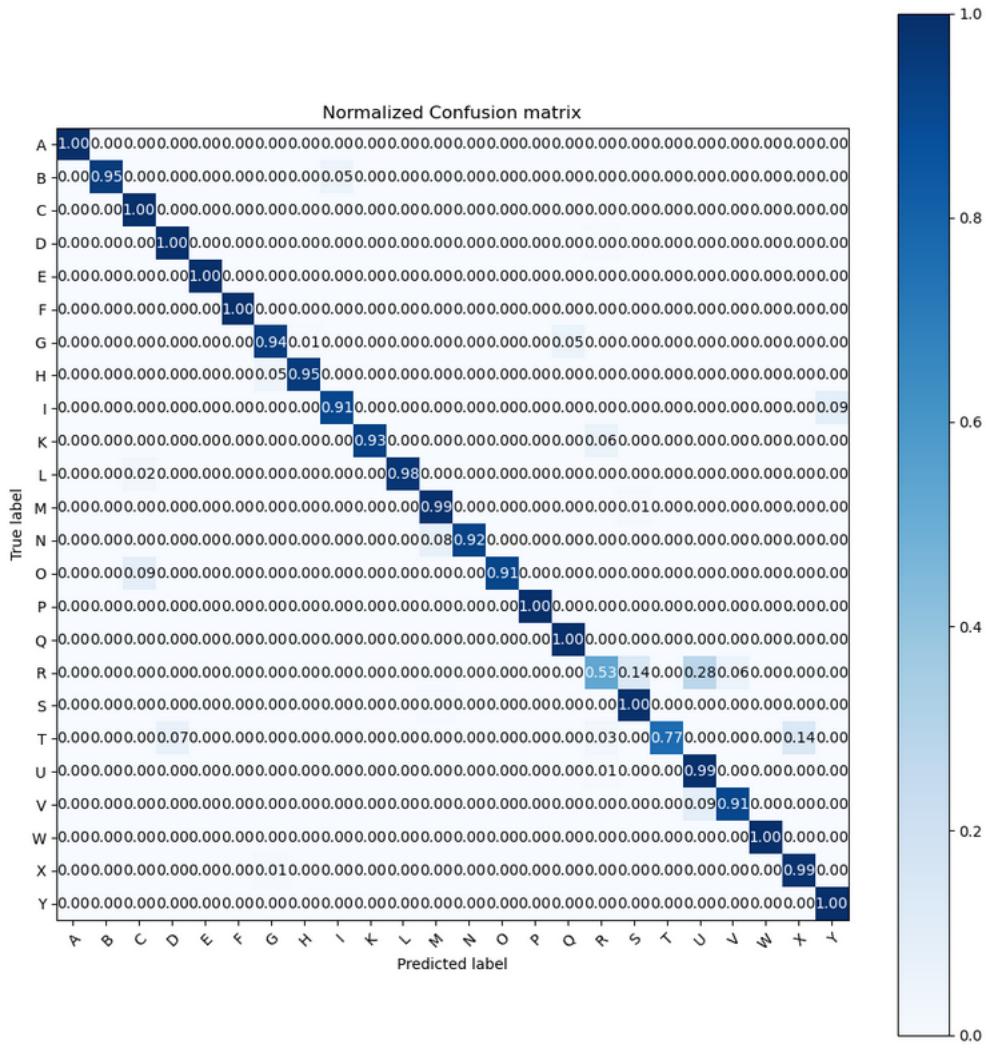


Figura 11: Modelo 2 (CNN) - Normalized Confusion matrix.

3.5.5 Modelo CNN 5

Este modelo foi treinado para 50 epochs e com dropout de 0.5.

De seguida, avaliou-se o modelo usando o conjunto de dados de teste e obteve-se um valor de precisão de 0.899.

Depois do treino acabar, desenhou-se a variação de precisão com a época (figura 18).

De seguida, na figura 19 é apresentada a matriz de confusão normalizada e na figura 20 é apresentada a matriz de confusão não-normalizada.

A figura 21 corresponde à visualização das previsões dos rótulos das classes para algumas imagens de teste deste modelo.

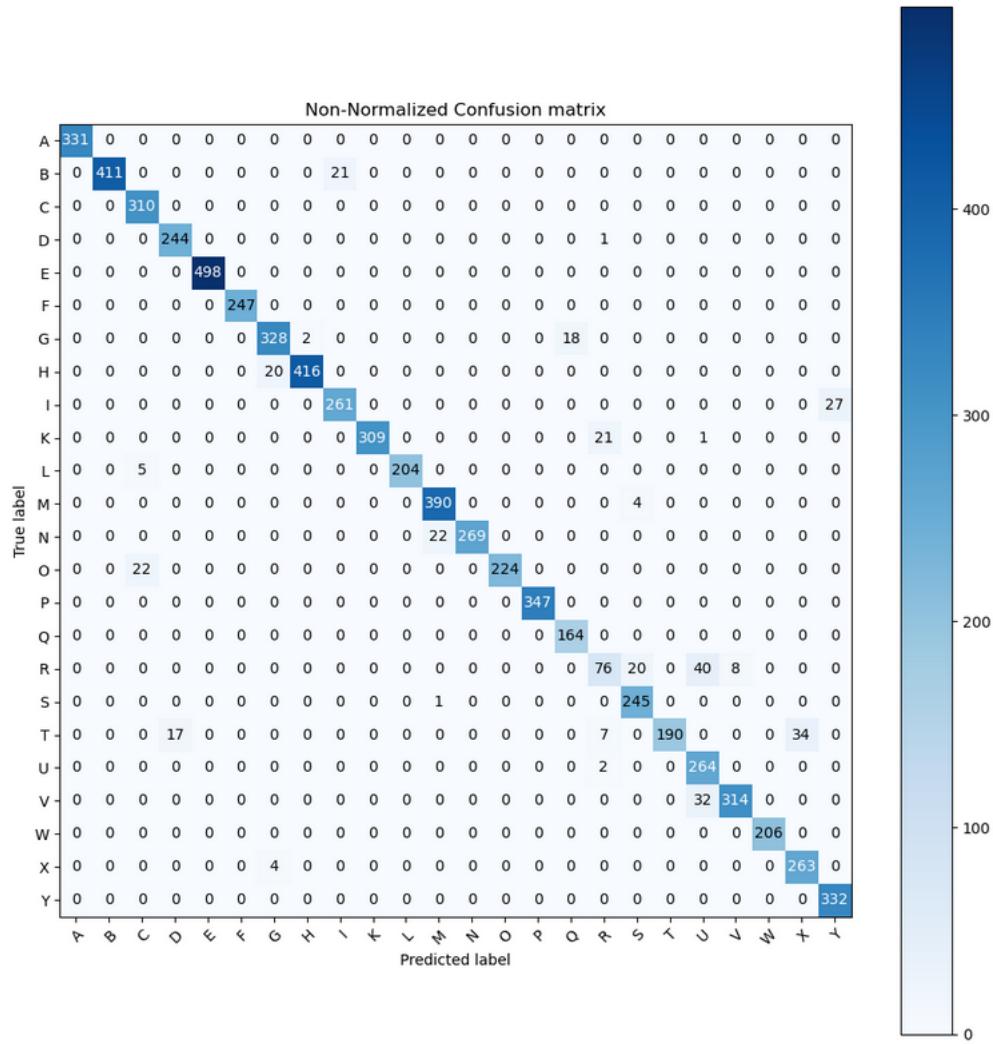


Figura 12: Modelo 2 (CNN) - Non-Normalized Confusion matrix.

4 Logistic Regression (LR)

4.1 Funcionamento do algoritmo LR

O LR é um algoritmo de ML utilizado para os problemas de classificação baseado no conceito de probabilidade, cujo objetivo é determinar a categoria/classe de saída.

O LR é um dos algoritmos de ML mais populares utilizado para os problemas de classificação, é um algoritmo de análise preditiva e baseado no conceito de probabilidade. Ou seja, é usado para prever a variável dependente usando um determinado conjunto de variáveis independentes.

A regressão logística prevê a saída de uma variável dependente categórica. Portanto, o resultado deve ser um valor categórico ou discreto. Pode ser '0' ou '1', mas em vez de fornecer o valor exato, fornece os valores probabilísticos entre 0 e 1.

Este algoritmo é muito semelhante à regressão linear, exceto na forma como são usadas. A



Figura 13: Modelo 2 (CNN) - Previsão da 25 imagens.

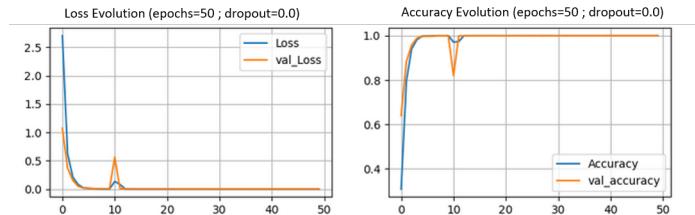


Figura 14: Modelo 4 (CNN) - Evolução da perda e Precisão.

regressão linear é usada para resolver problemas de regressão, enquanto a regressão logística é usada para resolver os problemas de classificação.

A regressão logística é um algoritmo de ML significativo porque tem a capacidade de fornecer probabilidades e classificar novos dados usando conjuntos de dados contínuos e discretos.

4.2 Processamento dos dados no modelo LR

Primeiramente, foram lidos os conjuntos de dados de treino e teste, como referido no subcapítulo 2.4.

De seguida, definiu-se os hiperparâmetros do modelo a implementar, tamanho do lote (*batch_size*) e taxa de aprendizagem (*learning_rate*). Esses hiperparâmetros controlam a ra-

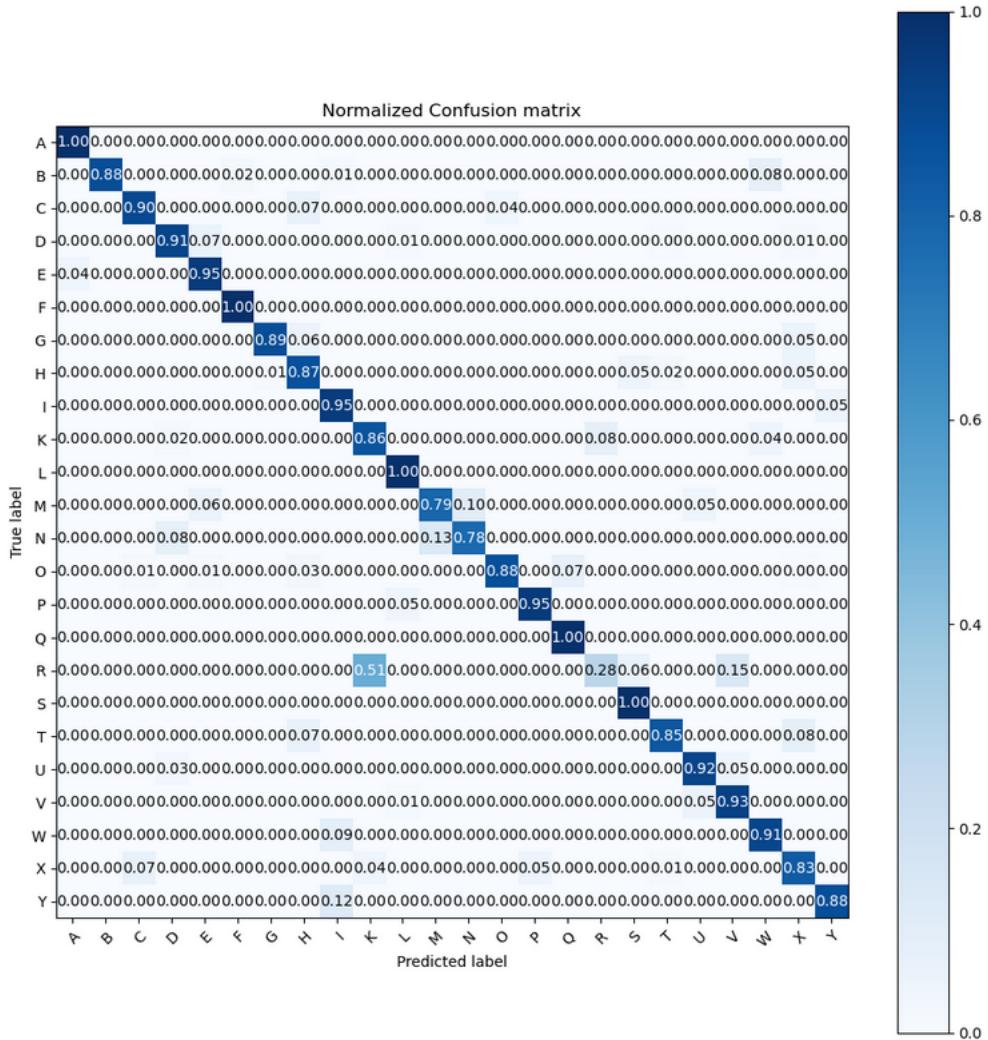


Figura 15: Modelo 4 (CNN) - Normalized Confusion matrix.

pidez com que o modelo aprenderá por meio dos casos de treino. O tamanho do lote determina quantas imagens o modelo carregará de cada vez. A taxa de aprendizagem determina o tamanho do ajuste dos parâmetros do modelo após cada fase de treino. Neste modelo, o tamanho do lote foi definido como 256 e a taxa de aprendizagem como 1e-5.

Foram definidas outras duas constantes: o tamanho da entrada (*input_size*) e o número de classes (*num_classes*). O tamanho de entrada é a quantidade de dados necessária para representar uma única imagem. Neste modelo, o tamanho de entrada é 784, pois a imagem é uma imagem em tons de cinza de 28x28 pixels (um valor para cada coordenada *x* e *y* correspondente) e o número de classes é 26, uma vez que os rótulos têm valores de 0 a 25, inclusive.

Depois de definir os hiperparâmetros, as matrizes de entrada de treino e teste foram convertidas para valores flutuantes contínuos para permitir uma aprendizagem mais precisa em comparação com valores discretos. Por outro lado, os rótulos de treino e teste foram convertidos para números inteiros longos, uma vez que a saída do modelo são índices a serem usados no acesso aos valores de probabilidade.

As matrizes de entrada e as matrizes de rótulo foram agrupadas num único objeto para os

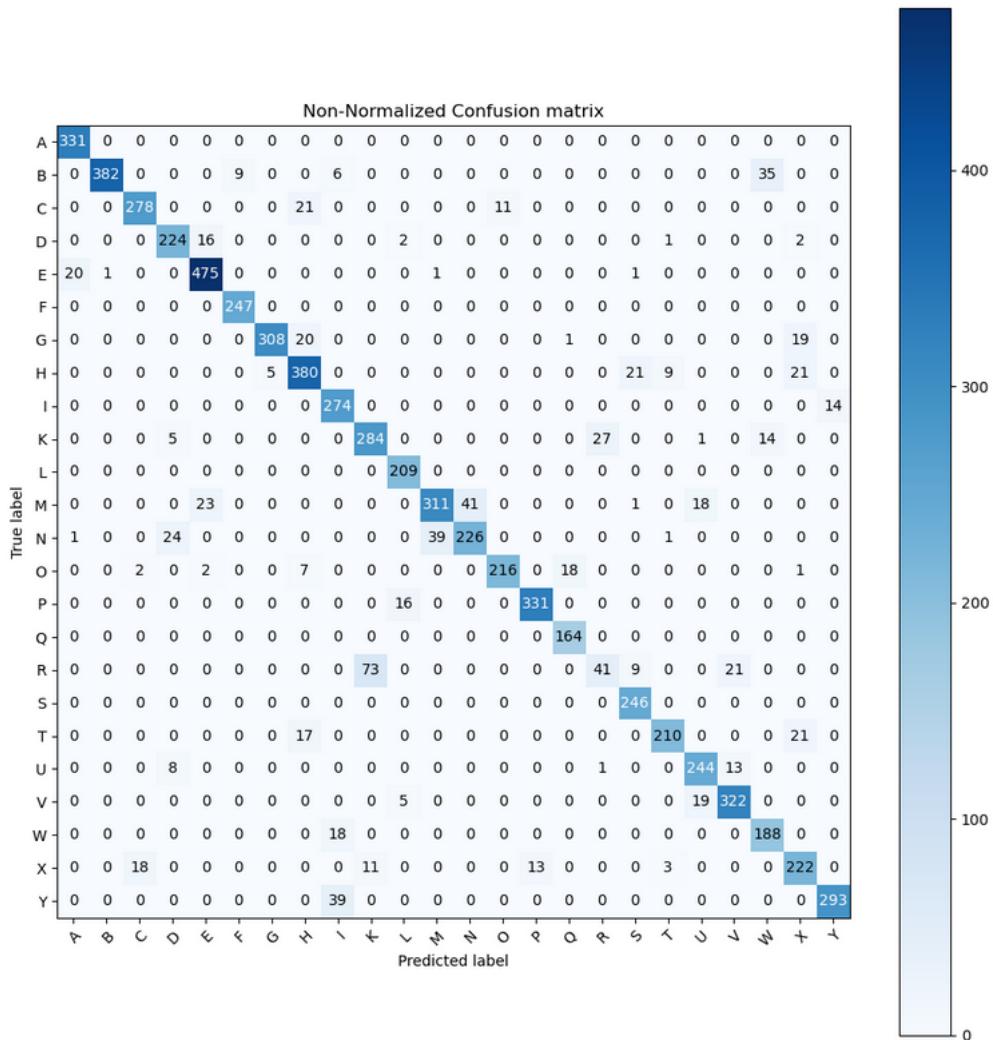


Figura 16: Modelo 4 (CNN) - Non-Normalized Confusion matrix.

conjuntos de dados de treino e teste. O conjunto de dados de treino também foi dividido num subconjunto de treino e validação. Neste projeto, 15% do conjunto de dados de treino original foi colocado no subconjunto de validação. Ficamos com um total de 23337 dados para treino e 4118 dados para validação.

4.3 Construção do modelo LR

Na regressão (função do *PyTorch*) *Linear*, cada variável independente é multiplicada por um peso e adicionada por algum desvio de compensação que afeta a classificação da saída. Nesse caso, as variáveis independentes são os valores dos pixels e a saída é a classificação das letras.

Inicialmente, a função retorna um tensor com 26 elementos com valores que variam de infinito negativo a infinito positivo, que, na verdade, não são infinitos, pois os valores são limitados pelo número de bits alocados na memória. Como tal, os valores devem ser normalizados entre



Figura 17: Modelo 4 (CNN) - Previsão da 25 imagens.

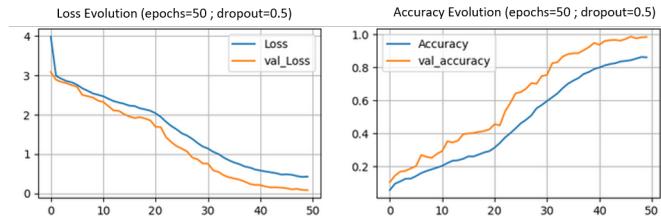


Figura 18: Modelo 5 (CNN) - Evolução da perda e Precisão.

0 e 1,0 inclusive.

A função *cross_entropy* de *PyTorch* foi usada para combinar a verossimilhança de log negativo e a função *softmax* para normalizar os valores resultantes da função linear. A função *softmax* é o termo dentro do logaritmo negativo. Cada valor normalizado do tensor é obtido substituindo cada valor pelo termo $\exp(x/\text{classe})$ dividindo pela soma de todos os exponenciais dos valores originais do tensor.

Ao instanciar a classe, são criados valores de peso aleatórios e valores de polarização, o que leva a criar os métodos *training_step* e *validation_step*. Por outro lado, os métodos *validation_epoch_end* e *epoch_end* existem para apresentar o desempenho atual do modelo.

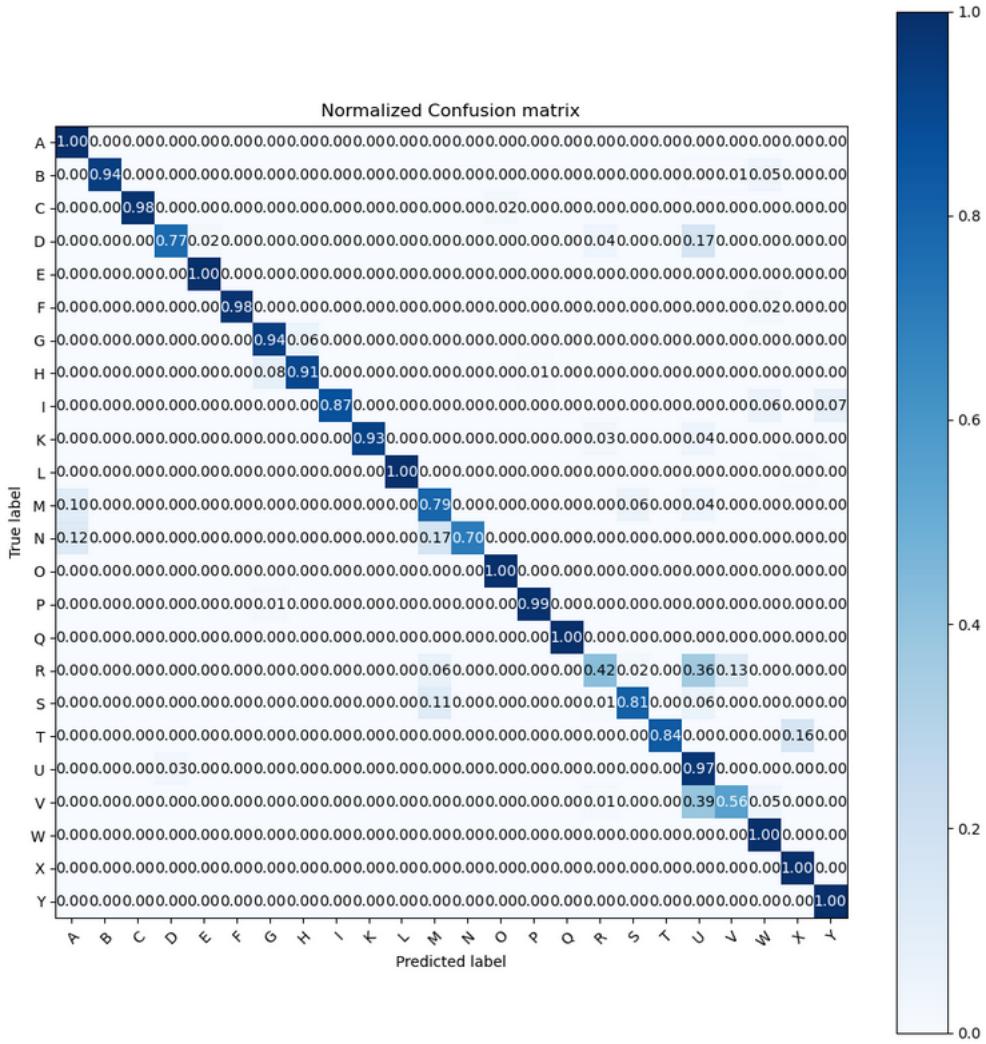


Figura 19: Modelo 5 (CNN) - Normalized Confusion matrix.

4.4 Treino do modelo LR

A maior parte da etapa de treino ocorre alimentando os lotes de conjunto de dados para o modelo. A função de precisão não seria capaz de determinar o melhoramento do modelo, uma vez que apenas olha para a saída. Como tal, usou-se o Stochastic Gradient Descent (SGD) para determinar o melhoramento do modelo.

A taxa de ajuste dos parâmetros iniciais para os novos parâmetros é ditada pela taxa de aprendizagem.

É importante chamar a função `zero_grad()` após cada ajuste dos parâmetros do modelo para redefinir os valores de gradiente para a função de perda.

Depois que a arquitetura da LR foi definida, procurou-se entender que fatores que afetam a precisão do modelo *LR*. Sendo assim, criaram-se quatro modelos para verificar se o número de *epochs* e se a taxa de aprendizagem afetam a precisão o modelo.

O modelo 1 foi treinado para 50 *epochs* (ou seja, 50 iterações) e com uma taxa de aprendizagem de 1e-4 e o modelo 2 foi treinado para 100 *epochs* e com com uma taxa de aprendizagem

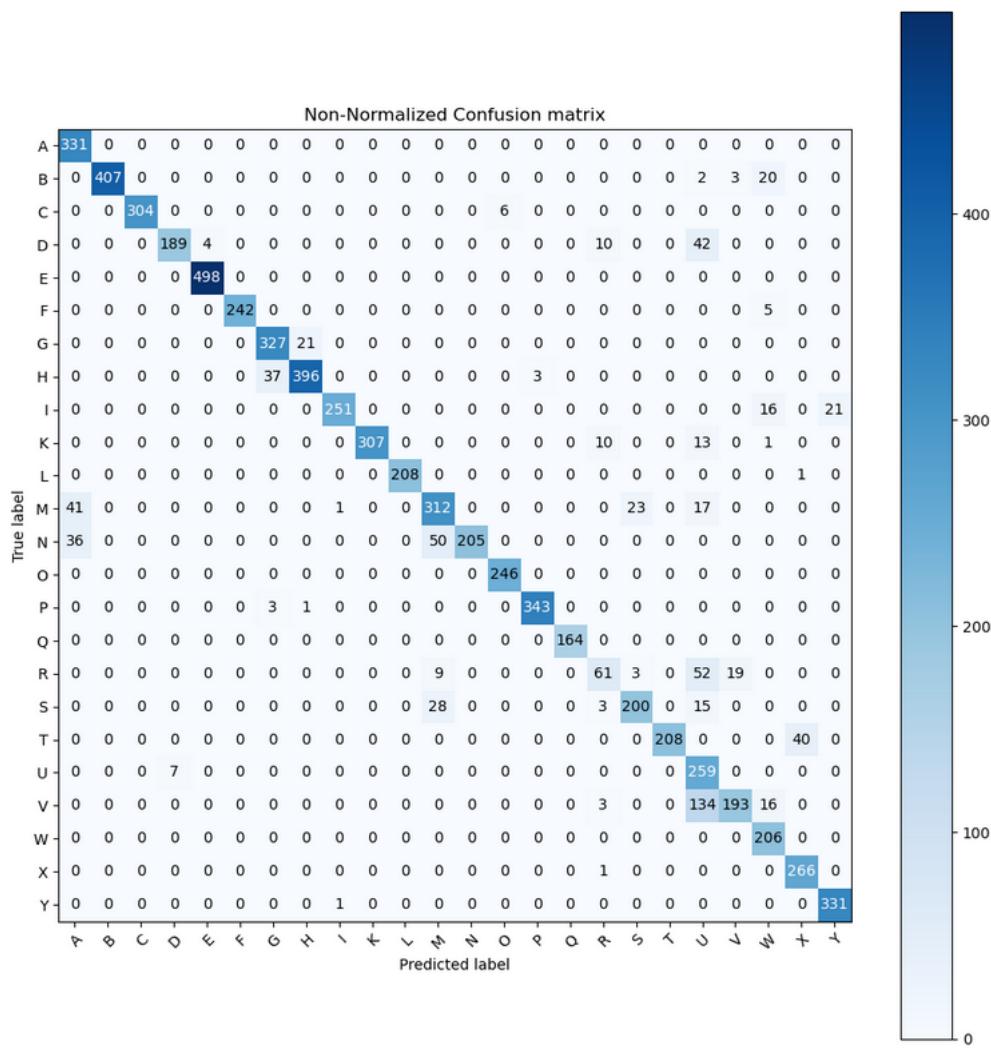


Figura 20: Modelo 5 (CNN) - Non-Normalized Confusion matrix.

de 1e-4. O modelo 3 foi treinado para 50 *epochs* e com uma taxa de aprendizagem de 1e-5 e o modelo 4 foi treinado para 100 *epochs* e com uma taxa de aprendizagem de 1e-5.

Os modelos 1 e 2, juntamente com os modelos 3 e 4, serviram para verificar qual o impacto do número de *epochs* no modelo LR. Os modelos 1 e 3, juntamente com os modelos 2 e 4, serviram para verificar qual o impacto da taxa de aprendizagem no modelo LR.

Depois de cada modelo LR pronto, foi preciso treiná-los alimentando o conjunto de treino.

4.5 Previsão do modelo LR

A fim de visualizar o resultado do treino, criou-se uma função auxiliar *predict_image()* para passar uma imagem para o modelo e retornar a previsão deste.

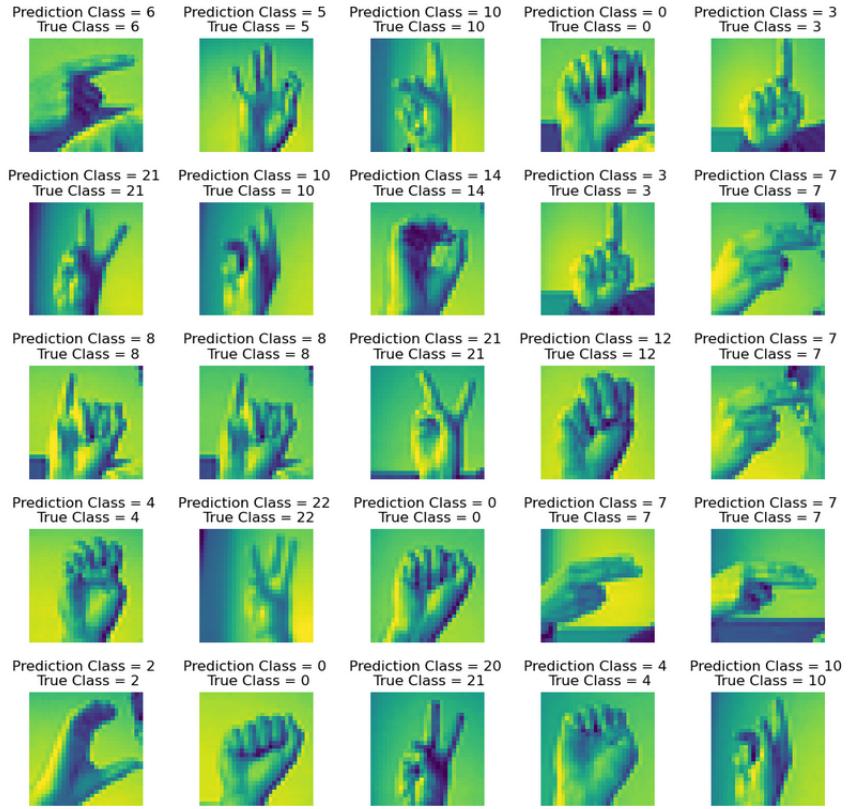


Figura 21: Modelo 5 (CNN) - Previsão da 25 imagens.

4.6 Desempenho do modelo LR

4.6.1 Modelo LR 1

Este modelo foi treinado para 50 *epochs* e com uma taxa de aprendizagem de 1e-4.

De seguida, avaliou-se o modelo usando o conjunto de dados de teste e obteve-se um valor de perda de 260.826 e um valor de precisão de 0.035. Como os parâmetros do modelo são aleatórios na instanciação, espera-se que a precisão seja muito baixa e a perda de validação seja alta.

Depois do treino acabar, desenhou-se a variação de precisão com a época (figura 22).

De seguida, avaliou-se o modelo usando o conjunto de dados de teste e obteve-se um valor de perda de 21.091 e um valor de precisão de 0.616.

4.6.2 Modelo LR 2

Este modelo foi treinado para 100 *epochs* e com uma taxa de aprendizagem de 1e-4.

Depois do treino acabar, desenhou-se a variação de precisão com a época (figura 23).

De seguida, avaliou-se o modelo usando o conjunto de dados de teste e obteve-se um valor

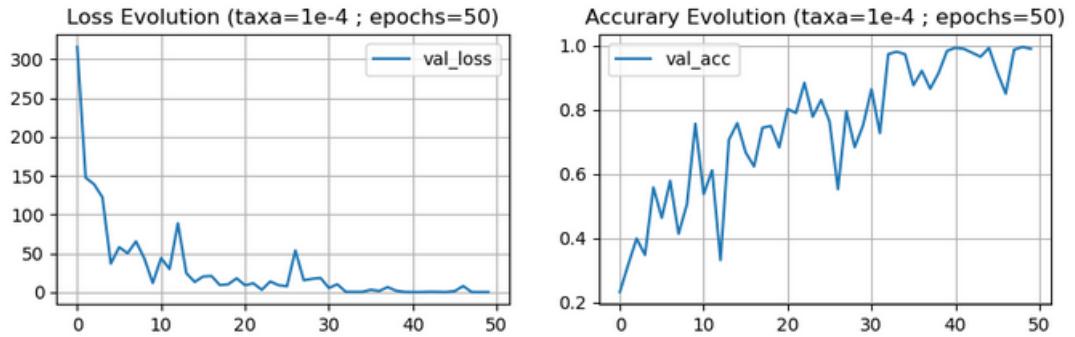


Figura 22: Modelo 1 (LR) - evolução de perda e de precisão.

de perda de 12.581 e um valor de precisão de 0.679.

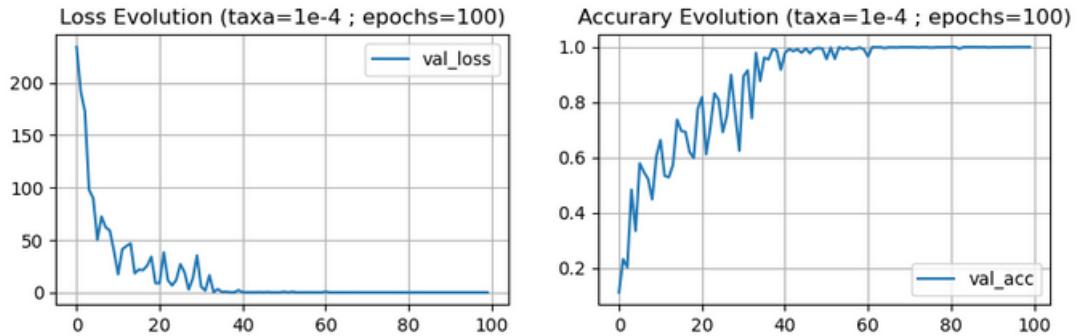


Figura 23: Modelo 2 (LR) - evolução de perda e de precisão.

4.6.3 Modelo LR 3

Este modelo foi treinado para 50 *epochs* e com uma taxa de aprendizagem de 1e-5.

Depois do treino acabar, desenhou-se a variação de precisão com a época (figura 24).

De seguida, avaliou-se o modelo usando o conjunto de dados de teste e obteve-se um valor de perda de 6.082 e um valor de precisão de 0.541.

4.6.4 Modelo LR 4

Este modelo foi treinado para 100 *epochs* e com uma taxa de aprendizagem de 1e-5.

Depois do treino acabar, desenhou-se a variação de precisão com a época (figura 25).

De seguida, avaliou-se o modelo usando o conjunto de dados de teste e obteve-se um valor de perda de 5.388 e um valor de precisão de 0.512.

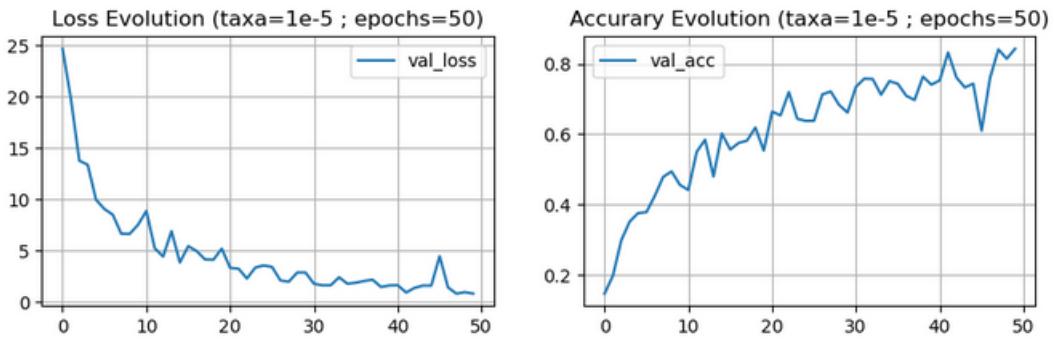


Figura 24: Modelo 3 (LR) - evolução de perda e de precisão.

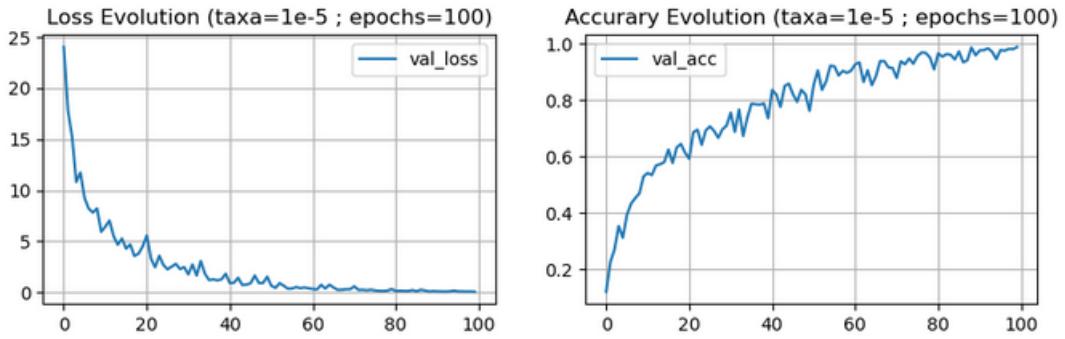


Figura 25: Modelo 4 (LR) - evolução de perda e de precisão.

5 Análise dos Resultados

Na Network Neural Convolucional (CNN), para chegarmos aos melhores resultados de previsão com a rede, variamos os *epochs* entre 20, 50 e 100 e o dropout entre 0.0, 0.2 e 0.5. Com estas variações concluímos que quanto mais *epochs* melhor será a precisão do modelo, mas mais de 50 *epochs* a percentagem de precisão do modelo tende a estagnar.

Os *epochs* são importantes para não termos um modelo *underfitting*, mas se usarmos demais, o nosso modelo torna-se *overfitting*, ou seja, um fenômeno quando o modelo tem um desempenho muito bom nos dados de treino, mas falha miseravelmente com os dados de teste.

Quanto os valores de *dropout*, se este valor for 0.0, temos um modelo *underfit*, mas se aumentarmos demasiado ocorre o problema de *overfitting*. Por isso, concluímos que para termos um modelo ideal o melhor valor para o dropout é 0.2.

O modelo 3 com 100 *epochs* e um *dropout* de 0.2, foi o modelo com a melhor precisão obtida, 95%.

No modelo de Regressão Logística (LR), para a obtenção da melhor rede de previsão, foram variados os valores dos *epochs* e da taxa de aprendizagem. Os valores dos *epochs* foram variados entre 50 e 100, ao que concluímos que quanto mais *epochs* melhor será a previsão da rede.

A partir de 100 *epochs* o modelo tende a estagnar na percentagem de previsão. Em relação à taxa de aprendizagem foi variada em 1e-4 e 1e-5, o que concluímos que quanto maior a taxa de aprendizagem melhor e mais rápida será a aprendizagem da rede. O modelo 2 com 100 *epochs* e uma taxa de aprendizagem de 1e-5, foi o modelo com a melhor precisão obtida, 68%.

Em relação aos 2 modelos podemos concluir que a Network Neural Convolucional (CNN) apresentou um desempenho destacado na classificação de imagens de símbolos em língua de sinais. A pontuação média de precisão do modelo é um pouco mais de 95% e pode ser melhorada ajustando os hiperparâmetros. A precisão pode ser melhorada se tivermos mais *epochs* de treino. No entanto, mais de 95% de precisão também é uma conquista.

6 Conclusões

Neste projeto foram implementadas todas as funcionalidades previstas e também foram implementados dois algoritmos abordados nas aulas teórico-práticas da unidade curricular.

Os resultados foram os esperados, mas é evidente que a imagem não é clara devido à sua pequena resolução. Isso pode ter afetado a precisão dos modelos implementados. Uma limitação adicional deste modelo foi sua incapacidade de reconhecer sinais em movimento, como as letras “J” e “Z”.

Para trabalhos futuros, poderia usar-se imagens com resolução mais alta que permitam que outros detalhes sejam extraídos das imagens. E também poderia adicionar-se a funcionalidade para reconhecer sinais dinâmicos, de forma a obter um sistema completo, para todas as letras do alfabeto.

Contribuição dos alunos

De forma a não sobrecarregar nenhum dos alunos, a distribuição do trabalho foi uniforme e o mais justa possível.

Assim sendo, pode-se dizer que a percentagem de contribuição de cada aluno foi de cinquenta por cento.

Referências

https://www.sas.com/pt_br/insights/analytics/neural-networks.html
<https://towardsdatascience.com/introduction-to-machine-learning-algorithms-linear-regression-14c4e325882a>
<https://machinelearningmastery.com/linear-regression-for-machine-learning/>
<https://wiki.pathmind.com/neural-network>
<https://icrowdnewswire.com/2017/12/11/diferenca-entre-classificacao-e-regras-na-aprendizagem-de-maquinas/>
<https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148>
<https://stackoverflow.com/questions/4752626/epoch-vs-iteration-when-training-neural-networks?fbclid=IwAR0DJB1mdLQ3xcKTIV-NMwKbRfoLV7vPf6HEpSgXvKYuENcljBMtKL9VkKs>
<https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/>
<https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9>