Análise de Técnicas Recentes em Classificação de Imagens

João Brito, M9984
Department of Computer Science
University of Beira Interior
Covilhã, Portugal
joao.pedro.brito{at}ubi.pt

Ricardo Domingos, M10259
Department of Computer Science
University of Beira Interior
Covilhã, Portugal
ricardo.domingos{at}ubi.pt

***Abstract*—O domínio de classificação de imagens caracteriza-se por ser uma das sub-áreas de estudo, dentro da Inteligência Artificial, com mais literatura e avanços. Deste modo, o presente documento visa a análise e discussão de dois artigos científicos publicados recentemente e enquadrados na área anteriormente mencionada.**

I. Introdução

Na última década, inúmeros avanços surgiram no campo da Visão Computacional, com um claro foco em classificação de imagens. Esta tarefa resume-se a introduzir uma imagem de um objeto/pessoa/animal num modelo e obter o seu *output* (classe a que supostamente pertence essa imagem). O advento das CNNs - redes neuronais especializadas em reconhecimento de padrões/caraterísticas em imagens - foi possível através do uso das diversas técnicas propostas ao longo dos anos. De um modo transversal, o componente basilar das redes neuronais é o neurónio artificial.

Na sua forma mais básica, realiza as seguintes operações: transforma linearmente os *inputs* recebidos com o auxílio de pesos configuráveis e promove não-linearidade ao aplicar uma função de ativação a esse valor, de modo a determinar o *output* (ativação do neurónio).

Com efeito, as duas técnicas abordadas neste documento representam esforços no sentido de criar novas funções de ativação dotadas de propriedades desejáveis e melhorar a atribuição/atualização de pesos.

II. Conjuntos de Dados

Ambos os artigos estudados basearam as suas experiências e resultados nas duas versões do *dataset* CIFAR [1], cujas caraterísticas se apresentam de seguida.

A. CIFAR-10

Esta versão do *dataset* conta com 60000 imagens RGB (32x32 píxeis). Como o nome indica, estão presentes 10 classes mutuamente exclusivas e igualmente distribuídas (6000 imagens por classe). As referidas classes ilustram, essencialmente, espécies de animais e veículos de várias categorias. Por fim, a divisão treino-teste foi feita num rácio 5:1, ou seja, 50000 imagens para treino e 10000 imagens para teste.

[1]https://www.cs.toronto.edu/ kriz/cifar.html

B. CIFAR-100

A versão com 100 classes do CIFAR contém 600 imagens por classe (totalizando 60000 instâncias únicas), segue a temática de animais e veículos e o rácio treino-teste mantém-se. As 100 classes estão ainda agrupadas em 20 superclasses (menos específicas).

III. Métricas de Avaliação

Os artigos que foram alvo de estudo para o presente projeto, fazem uso da *accuracy* como métrica principal para avaliar o resultado das suas experiências. Esta medida de avaliação apresenta a seguinte fórmula:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

Analisando a fórmula 1, importa notar que os subgrupos TP, TN, FP e FN são mutuamente exclusivos e representam os *outputs* do modelo: T e F indicam, respetivamente, se a classificação foi correta ou não, quando comparada aos rótulos de base (*ground-truth*); P e N referem-se à associação de uma imagem a uma dada classe. Pode-se, ainda, perceber que o denominador representa todas as classificações feitas pelo modelo em uso, enquanto que o numerador contém os resultados que realmente estão corretos. Deste modo, se o modelo só fez classificações corretas (TP + TN = TP + TN + FP + FN = comprimento do *dataset*), então teve um desempenho excelente, formalizado com uma *accuracy* de 1 (ou 100%).

IV. Função de Ativação Mish

Existem várias funções de ativação largamente utilizadas pela comunidade científica, das quais se destacam: ReLU (e as suas revisões, Leaky ReLU e Parametrized ReLU), Sigmóide, TanH e, mais recentemente, Swish [2]. Assim, uma nova função de ativação terá de ser capaz de igualar ou superar o estado da arte e ser fácil de implementar. Para tal, foi proposta a função Mish [1], cujas caraterísticas serão enunciadas ao longo das secções seguintes.

A. *Descrição e Propriedades da Função Mish*

A função Mish é definida com recurso a duas outras funções bem conhecidas, nomeadamente, a TanH e a Softplus:

$$mish(x) = x \cdot tanh(softplus(x)) \tag{2}$$

A equação 2 pode ser desdobrada fazendo uso das definições da TanH e da Softplus:

$$mish(x) = x \cdot \frac{2}{1 + e^{-2\cdot(ln(1+e^{x}))}} - 1 \tag{3}$$

Devido às suas caraterísticas e comportamento, a função Mish pode-se comparar com a ReLU e a Swish. Os gráficos 1 e 2 apresentam cada uma das três funções referidas, bem como as suas derivadas (cruciais para a atualização de pesos com a descida do gradiente).

Através da análise visual e formal da função Mish, podem-se derivar as seguintes propriedades:

- **Não saturada:** na parte positiva do gráfico a função é monótona crescente (não limitada superiormente), o que permite manter informação de gradiente para a fase de atualização de pesos;
- **Suave e contínua**: ao contrário da ReLU, esta nova função não apresenta pontos de descontinuidade nem pontos angulosos, locais onde a derivada seria indefinida.
- **Diferenciável ao longo de todo o domínio:** Como consequência da propriedade anterior, a função Mish admite sempre derivada;
- **Ativação para *inputs* negativos pequenos:** de modo semelhante à Swish, o facto de alguns *inputs* negativos produzirem ativações permite manter a expressividade do modelo em uso.
- **Regularização intrínseca:** De]-inf,-1.1924] a função tende para 0, descartando, na prática, valores demasiado grandes em valor absoluto. Tal capacidade é eficaz na prevenção do problema da explosão de pesos.

De um modo geral, a Mish e a Swish partilham bastantes caraterísticas e, inclusive, possuem curvas bastante semelhantes. Ambas as funções referidas procuram superar a função de ativação por defeito em muitas arquiteturas existentes: a ReLU. Tal manifesta-se essencialmente de duas formas: a suavidade e as ativações para valores negativos.

Por um lado, a suavidade permite que o *output landscape* - superfície composta pelas saídas de uma rede com determinada função de ativação - não apresente mudanças bruscas de valor ou de comportamento, tal como sucede com a ReLU (figura 3). Sabendo que a função de custo recebe como argumento a saída do modelo, pode-se derivar uma relação de dependência entre as superfícies da função de custo e dos *outputs* do modelo. Naturalmente, a suavidade e distribuição de uma (*outputs*) manifesta-se na outra (custo), existindo claros benefícios em realizar a descida do gradiente numa superfície com menos irregularidades [2].

Por outro lado, tanto a Mish como a Swish preservam ativações para valores negativos relativamente pequenos, podendo esta informação ser útil na generalização dos dados. Adicionalmente, esta caraterística contrasta com a observada na ReLU, que mapeia qualquer valor negativo para 0. Como consequência, surge o grande problema da ReLU: a morte/desativação de neurónios [3].

Essencialmente, esta situação ocorre quando um ou mais neurónios produzem valores de saída iguais a 0, ou seja, não se ativam. Visto que a derivada da ReLU é negativa para valores negativos, os pesos desse neurónio não são atualizados. Como o algoritmo de retropropagação segue das últimas camadas para as primeiras, os neurónios que antecedem e estão ligados ao neurónio desativado, também podem ficar desativados. Contudo, esta situação não se verifica sempre, existindo formas de minimizar a sua ocorrência, tais como: *mini-batch/batch learning*, Leaky ReLU/Parametrized ReLU, *batch normalization* [4] ou *dropout* [5].

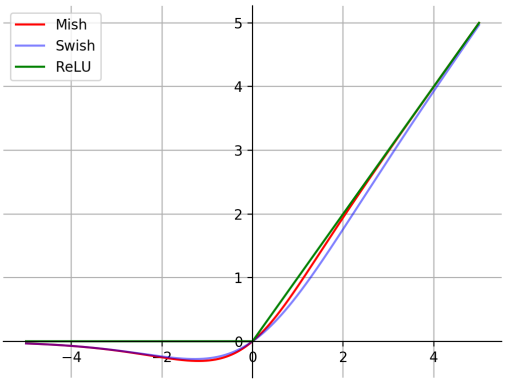

Fig. 1: Comparação entre as funções Mish, Swish e ReLU

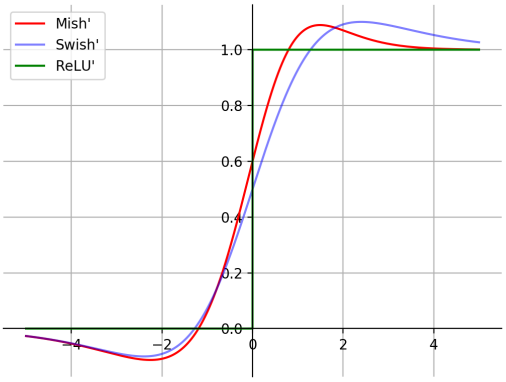

Fig. 2: Comparação entre as derivadas das funções Mish, Swish e ReLU

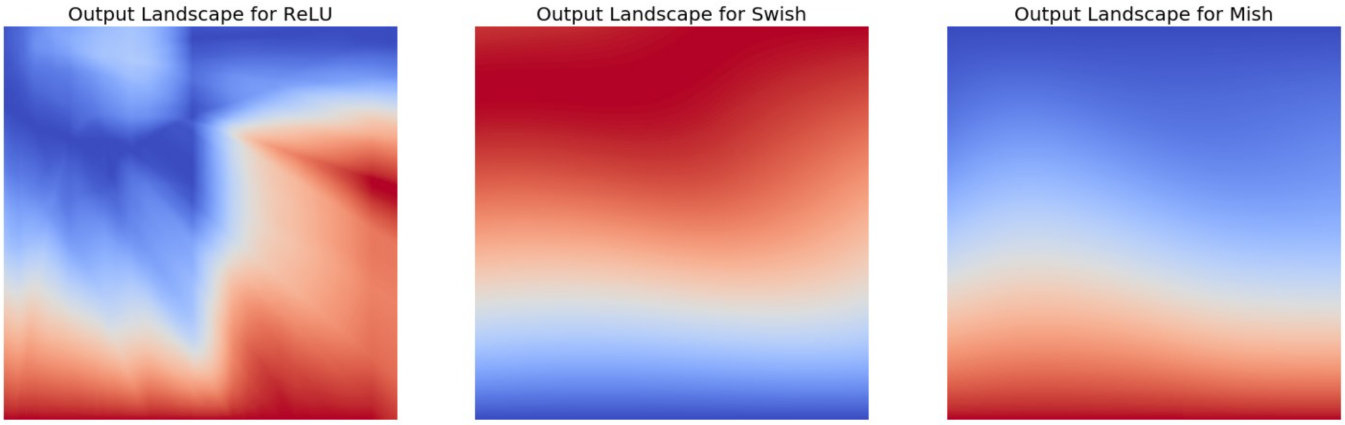

Fig. 3: Visualização do *output landscape* das funções ReLU, Mish e Swish. As figuras foram geradas usando uma rede neuronal com duas entradas (coordenadas (x,y)), uma saída e cinco camadas, cujos pesos foram aleatoriamente inicializados.

V. SWA: Stochastic Weight Averaging

Tradicionalmente o método mais usado para a otimização de modelos em *machine learning* é a SGD. No entanto, mesmo sendo este um dos métodos mais usados existem pontos onde pode ser melhorado. O método do SWA, vem tentar melhorar um pouco os resultados da técnica da SGD, sendo construída através da média de vários pontos na trajetória da SGD.

A ideia da construção do método da SWA, vem de uma observação empírica de que o mínimo local encontrado no final de cada ciclo tendia a ficar na periferia da área onde a percentagem de erro no teste era menor na superfície do erro. Fazendo a média de alguns desses pontos é possível obter melhor generalização (ver o primeiro gráfico da figura 4).

Na figura 4, no segundo e no terceiro gráfico estão representados os pesos da SGD e da SWA para 125 épocas nas superfícies erro de teste e de *loss* do treino respetivamente. Destes gráficos podemos concluir que enquanto a SGD procura diminuir a *loss* do treino, a SWA procura diminuir o erro de teste, podendo assim fazer uma melhor generalização.

O método ainda estuda outra vertente, a taxa de aprendizagem, neste caso é utilizado uma taxa de aprendizagem cíclica ou constante permitindo assim ao modelo fazer *exploration*, isto difere do método da SGD porque tradicionalmente a taxa de aprendizagem varia linearmente ou é exponencialmente descendente, dando assim uma ideia inicial de *exploration* para depois passar a uma fase de *exploitation*.

A. Considerações sobre a Taxa de Aprendizagem

Neste estudo a taxa de aprendizagem funciona de duas maneiras diferentes, uma delas é cíclica onde em cada ciclo a taxa de aprendizagem decresce linearmente entre um intervalo $\alpha 1$ e $\alpha 2$ (ver figura 5). Este processo pode ser representado através das seguintes formulas:

- $\alpha(i) = (1 - t(i))\alpha 1 + t(i)\alpha 2$,
- $t(i) = \frac{1}{c}(mod(i-1, c) + 1)$.

Neste caso, a taxa de aprendizagem é descontínua no final de cada ciclo, fazendo assim o salto da taxa de aprendizagem mínima para a máxima, ao contrário de outros métodos em que aumentam gradualmente.

Por outro lado, foi utilizada uma taxa de aprendizagem constante em todas as iterações, podendo-se representar da seguinte forma, $\alpha(i) = \alpha 1$

Estes métodos são utilizados porque neste estudo deram mais importância a fazer *exploration*, para o caso de querermos uma *exploration* mais alta poderemos usar a taxa de aprendizagem constante.

Por norma a taxa de aprendizagem cíclica obtém melhores resultados de *accuracy* do que a taxa de aprendizagem constante. Isto pode ser explicado porque com a taxa de aprendizagem cíclica, o algoritmo passa a fazer *fine-tunning* com a taxa de aprendizagem a decrescer, por outro lado com uma taxa de aprendizagem constante ao dar passos mais largos consegue ter uma maior *exploration*, no entanto acaba por ter piores resultados.

B. Algoritmo SWA

Algorithm 1 Algoritmo SWA

Require: $pesos\ \widehat{w}, valores\ da\ taxa\ de\ aprendizagem\ \alpha 1\ e\ \alpha 2, tamanho\ do\ ciclo\ c\ (para\ uma\ taxa\ de\ aprendizagem\ constante\ c = 1),\ número\ de\ iterações\ n$

Ensure: w_{SWA}

$w \leftarrow \widehat{w}$

$w_{SWA} \leftarrow w$

for $i \leftarrow 1, 2, ..., n$ **do**

  $\alpha \leftarrow \alpha(i)$

  $w \leftarrow w - \alpha \bigtriangledown \mathcal{L}_I(w)$

  if mod(i,c) = 0 **then**

    $n_{models} \leftarrow i/c$

    $w_{SWA} \leftarrow \frac{w_{SWA} \cdot n_{models} + w}{n_{models}+1}$

  end if

end for

Este algoritmo, tem muitas semelhanças ao algoritmo da SGD. Inicialmente recebe se pesos já pré treinados, por norma

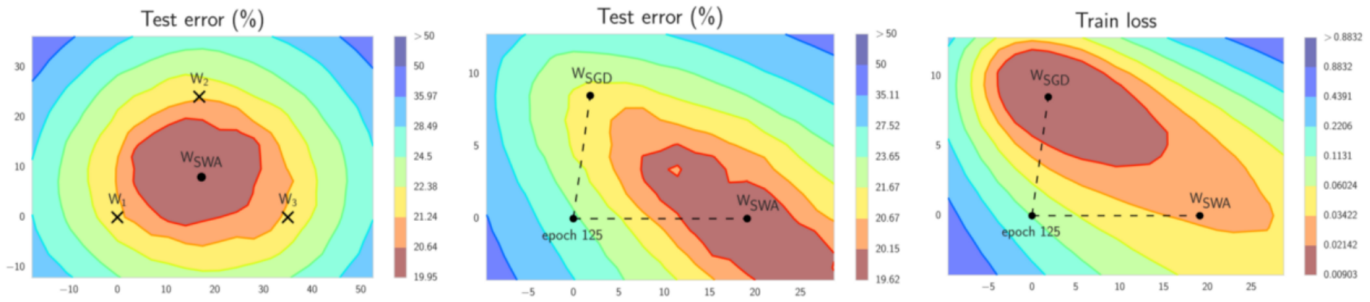

Fig. 4: 1º gráfico: Estão representados 3 pesos de redes treinadas com SGD: W_1, W_2, W_3 e W_{SWA}; 2º gráfico: Pesos finais da SWA e da SGD na superfície do erro de teste. 3º gráfico: Pesos finais da SWA e da SGD na superfície da *loss* de treino.

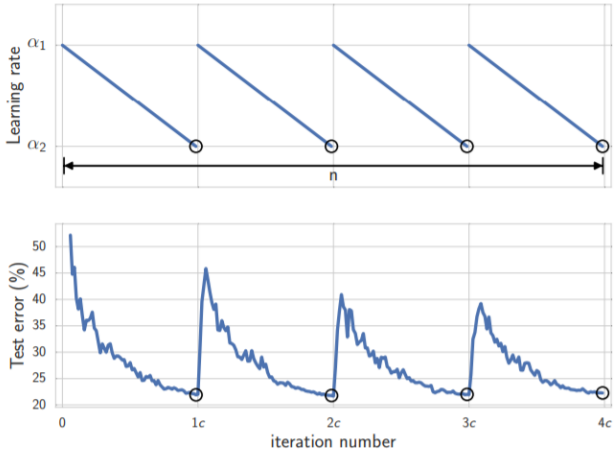

Fig. 5: Comportamento da taxa de aprendizagem cíclica.

com o algoritmo da SGD, recebem-se ainda dois valores ($\alpha 1$ e $\alpha 2$) que definem o mínimo e o máximo do intervalo do *learning rate*. Define se c, um determinado número de iterações que cada ciclo terá. Por fim define se o número de iterações que o algoritmo correrá.

Após estas variaveis estarem definidas, inicializa-se os pesos, tanto da SGD como da SWA, com os pesos já pré treinados. De seguida durante todas as iterações n, α será atualizado consoante a iteração e, após este passo, é a vez dos pesos serem atualizado segundo a fórmula da SGD.

Neste passo, durante as iterações n, em cada conclusão do ciclo c, é guardado o nímero de ciclos percorridos, este valor é depois aplicado na fórmula da atualização dos pesos da SWA. Esta fórmula multiplica o número de ciclos aos pesos antigos do SWA, depois somando os pesos da SGD da última iteração, este resultado é dividido pelo número de ciclos mais 1.

C. Complexidade computacional

Em relação a recursos utilizados, o método do SWA utiliza praticamente os mesmos recursos. A nível de memória, apenas será necessário guardar os pesos de uma rede neuronal. Já a nível de tempo, terá apenas mais o tempo de fazer a média dos pesos, que é praticamente insignificante. Em resumo, como os dois têm processos semelhantes, por consequência, utilizam praticamente os mesmos recursos.

VI. Análise Comparativa

O presente capítulo procurará fornecer provas empíricas para as observações teóricas até aqui discutidas. De notar que as tabelas referidas ao longo das secções seguintes se encontram na última página. Nestas, qualquer valor que surja enfatizado a negrito é o melhor, dentro do seu contexto.

O computador usado para realizar os testes, está equipado com, entre outros componentes, uma *GPU* RTX 2080 Ti.

A. Mish: Testes e Resultados

Pelos testes efetuados, foi possível replicar, até certo ponto, os resultados observados no *paper* original. Da tabela I podem-se extrair várias informações. Por um lado, o modelo Inception-V3 consome, aproximadamente, o dobro de recursos (mais precisamente, tempo) que a DenseNet-121, com ganhos na ordem de 1%, pelos nossos testes. Por outro lado, as três funções de ativação estão bem equilibradas em termos de *accuracy*, não existindo grande fator de separação entre elas.

B. SWA: Testes e Resultados

O método foi testado tanto no CIFAR-10 como no CIFAR-100. Para ambas foram utilizados os seguintes modelos: VGG-16, Pre-ResNet 164 e Wide ResNet-28-10. Além destes modelos foram ainda utilizados a ShakeShake-2x64d para o CIFAR-10 e a PyramidNet-272 para o CIFAR-100.

Para cada modelo foi definido o *budget* como o número de épocas necessárias para treinar o modelo até convergir, com a SGD. Os modelos VGG, Pre-ResNet e Wide ResNet são treinados com a SGD até 75% do *bugget* de treino, servindo os pesos da última época para inicializar os pesos da SWA. Para os testes da SWA, foi executada para 0.25, 0.5 e 0.75 do *budget* para completar 1, 1.25 e 1.5 *budget* respetivamente. Para os outros dois modelos utilizados a SWA é inicializada no final do *budget*. Para tal só existem resultados para 1.25 e 1.5 do *budget* executando a SWA apenas 0.25 e 0.5 do *budget*.

Para cada resultado apresentado na tabela II foi calculada uma média de 3 execuções e o seu respetivo desvio padrão.

Da tabela II podemos retirar a conclusão que a SWA tem uma performance melhor que a SGD: cerca de 0.5% no CIFAR-10 e 0.75%-1.5% no CIFAR-100.

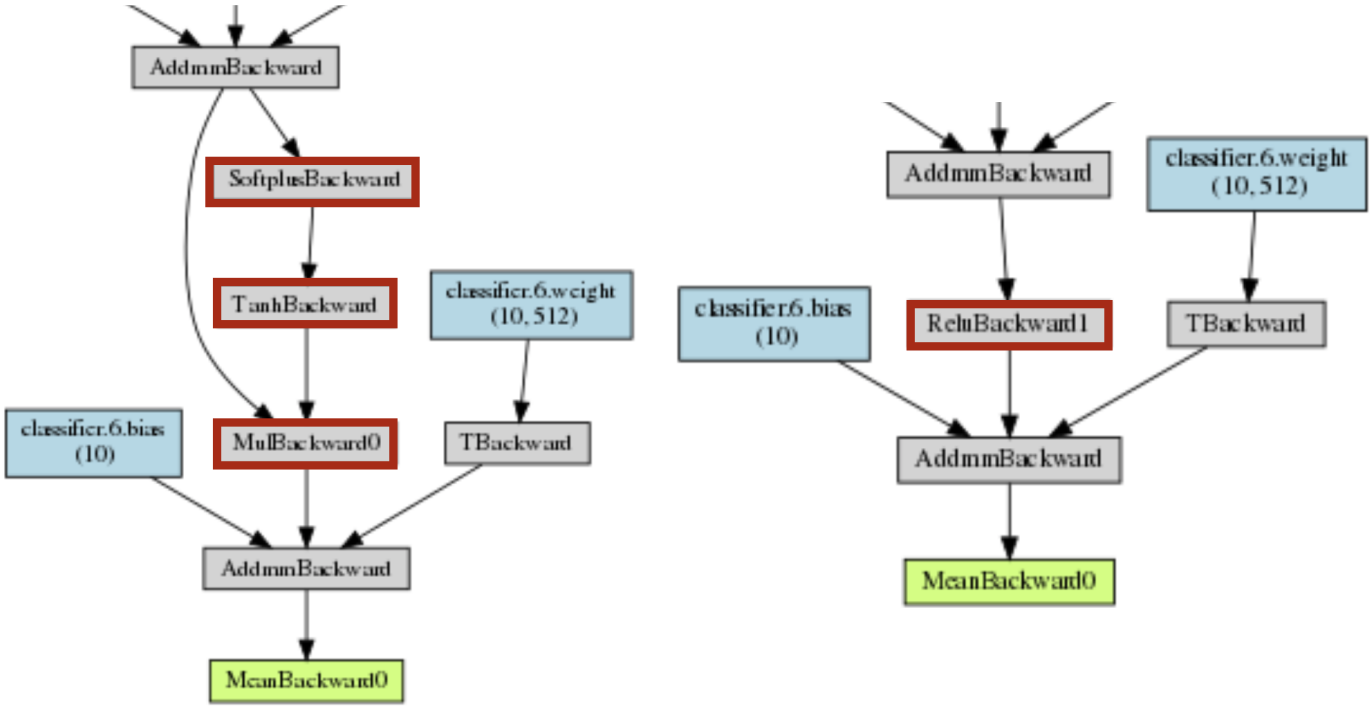

Fig. 6: Porção da arquitetura da VGG-16 na sua versão modificada (esquerda) e original (direita). A versão modificada conta com a função Mish como a sua última função de ativação (sendo que os nós a vermelho incorporam a fórmula 2), ao passo que a original usa apenas a ReLU.

C. Complementação das Duas Técnicas

A título experimental, procurou-se conjugar os dois métodos analisados, de forma a melhorar os resultados obtidos individualmente. Para tal, juntou-se a Mish à implementação do SWA, devido ao nível de personalização dos modelos deste. Assim, usou-se a VGG-16, pela sua rapidez de execução, permitindo a realização de um maior número de testes. Do ponto de vista prático, as alterações ao modelo original restringiram-se à substituição da última instância da ReLU (figura 6). Esta metodologia foi adotada com vista a replicar a implementação associada ao *paper* da função Mish [2] (i.e. tomando como exemplo a DenseNet-121, apenas foi trocada a última instância da ReLU pela Mish).

Seguindo as experiências anteriores, definiram-se 200 épocas de treino, sendo que o único parâmetro alterado foi a taxa de aprendizagem. Finalmente, os resultados mostrados dizem respeito à média e desvio padrão de três execuções idênticas. A tabela IV começa por apresentar as duas taxas de aprendizagem por defeito: 0.05 para SGD e 0.01 para SWA. Com esta configuração, a ReLU surge na frente, ainda que com uma margem mínima. De seguida, procurou-se diminuir o passo de aprendizagem, fruto das caraterísticas da função Mish (o artigo original fez alusão à vantagem aparente desta função quando aliada a taxas de aprendizagem mais baixas). Seria, então, de esperar que a diferença entre a ReLU e a Mish fosse mais reduzida.

[2]https://github.com/digantamisra98/Mish

Por fim, e de forma intermédia, utilizaram-se taxas de aprendizagem iguais a 0.01. Esta última configuração, deu a primeira liderança à função Mish, ainda que com uma margem novamente diminuta (semelhante à diferença registada na primeira configuração).

Os testes efetuados parecem indicar a inexistência de uma superioridade clara de uma função em relação à outra. Por outro lado, com uma maior variedade de testes (e.g. modelos mais profundos, mais variações de taxa de aprendizagem, número de épocas, entre outros) poder-se-ia derivar uma conclusão mais fundamentada.

Numa nota adicional, os resultados na primeira e terceira configurações da tabela IV para a Mish, são superiores a qualquer resultado reportado no *paper* da Mish, no CIFAR-10. Repare-se que o modelo VGG-16 não foi utilizado no *paper* da Mish e as implementações podem variar com relação ao *paper* do SWA. Estas observações podem ser justificados com os detalhes referidos, não deixando, contudo, de ser interessantes.

VII. Conclusão

O projeto descrito neste documento visou a análise de duas publicações científicas publicadas nos últimos anos. Por um lado, a função Mish carateriza-se por possuir propriedades desejáveis e que a colocam num nível semelhante à ReLU (função por defeito para muitas aplicações). Ainda assim, o aspeto onde a ReLU parece ter vantagem é no tempo e custo computacional necessários para treinar o modelo associado (tabela IV e fórmula 3).

Por outro lado, o método SWA procura trazer benefícios em relação à SGD, que, mais uma vez, é a metodologia canónica no que toca à otimização de pesos.

Os testes permitiram-nos concluir que as inovações introduzidas pelas duas publicações melhoram, em certa medida, o estado da arte.

Por fim, a solução híbrida carece de experimentação e trabalho futuro, nomeadamente, no tipo de modelo utilizado, taxas de aprendizagem e arquitetura em geral.

REFERENCES

[1] Diganta Misra. (2019). Mish: A Self Regularized Non-Monotonic Neural Activation Function.

[2] Prajit Ramachandran, Barret Zoph & Quoc V. Le. (2017). Swish: A Self-Gated Activation Function.

[3] Lu Lu, Yeonjong Shin, Yanhui Su, & George Em Karniadakis. (2019). Dying ReLU and Initialization: Theory and Numerical Examples.

[4] Sergey Ioffe, Christian Szegedy. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.

[5] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research, 15(56), 1929-1958.

[6] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, Andrew Gordon Wilson. (2018). Averaging Weights Leads to Wider Optima and Better Generalization.

Modelo (Função de Ativação)	Resultado do *paper* (%)	Época Final - Replicado (%)	Melhor Época (%)	Tempo de Treino (s)
		CIFAR-10		
Inception-V3 (Mish)	91.19	91.71 ± 0.02	91.89 ± 0.14	18991.24 ± 15.29
DenseNet-121 (Mish)	91.27	90.72 ± 0.27	91.09 ± 0.10	9226.21 ± 46.01
DenseNet-121 (Swish)	90.92	90.68 ± 0.28	91.10 ± 0.00	9604.06 ± 355.49
DenseNet-121 (ReLU)	91.09	90.8 ± 0.09	91.19 ± 0.32	9590.13 ± 51.02
		CIFAR-100		
ResNext-50 (Mish)	67.58	68.19 ± 0.15	68.86 ± 0.19	12308.98 ± 6.71
DenseNet-121 (Mish)	66.31	65.94 ± 0.56	66.95 ± 0.26	9246.68 ± 22.69

TABLE I: Resultados replicados do *paper* sobre a função Mish [1].

		SWA		
Modelo (Budget)	SGD (%)	1 Budget (%)	1.25 Budget (%)	1.5 Budget (%)
		CIFAR-10		
VGG-16 (200)	93.25 ± 0.16	93.59 ± 0.16	93.70 ± 0.22	93.64 ± 0.18
ResNet-164 (150)	95.28 ± 0.10	95.56 ± 0.11	95.77 ± 0.04	95.83 ± 0.03
WRN-28-10 (200)	96.18 ± 0.11	96.45 ± 0.11	96.64 ± 0.08	96.79 ± 0.05
ShakeShake-2x64d (1800)	96.93 ± 0.10	–	97.16 ± 0.10	97.12 ± 0.06
		CIFAR-100		
VGG-16 (200)	72.55 ± 0.10	73.91 ± 0.12	74.17 ± 0.15	74.27 ± 0.25
ResNet-164 (150)	78.49 ± 0.36	79.77 ± 0.17	80.18 ± 0.23	80.35 ± 0.16
WRN-28-10 (200)	80.82 ± 0.23	81.46 ± 0.23	81.91 ± 0.27	82.15 ± 0.27
PyramidNet-272 (300)	83.41 ± 0.21	–	83.93 ± 0.18	84.16 ± 0.15

TABLE II: Resultados originais do *paper* sobre o SWA [6].

Modelo (Função de Ativação)	Resultado do *paper* (%)	Época Final - Replicado (%)	Melhor Época (%)	Tempo Médio de Treino (s)
		CIFAR-10		
VGG-16 (ReLU)	93.59 ± 0.16	93.53 ± 0.08	93.57 ± 0.08	1643.24 ± 7.73
Resnet-164 (ReLU)	95.56 ± 0.11	95.62 ± 0.05	95.71 ± 0.02	6968.53 ± 117.24
		CIFAR-100		
VGG-16 (ReLU)	73.91 ± 0.12	73.67 ± 0.25	73.67 ± 0.25	1642.23 ± 5.53
Resnet-164 (ReLU)	79.77 ± 0.17	79.53 ± 0.26	79.56 ± 0.29	6830.03 ± 2.78

TABLE III: Resultados replicados do *paper* sobre o SWA [6].

Função de Ativação	α SGD	α SWA	Época Final (%)	Melhor Época (%)	Tempo Médio de Treino (s)
ReLU	0.05	0.01	**93.43 ± 0.02**	**93.46 ± 0.002**	**1662.04 ± 11.28**
Mish	0.05	0.01	93.31 ± 0.18	93.35 ± 0.14	1662.24 ± 4.68
ReLU	0.001	0.001	**89.02 ± 0.09**	**89.02 ± 0.08**	**1654.35 ± 5.34**
Mish	0.001	0.001	88.59 ± 0.07	88.61 ± 0.05	1670.85 ± 1.80
ReLU	0.01	0.01	92.45 ± 0.16	92.50 ± 0.14	**1663.10 ± 2.99**
Mish	0.01	0.01	**92.56 ± 0.19**	**92.64 ± 0.21**	1745.10 ± 56.96

TABLE IV: Resultados da abordagem híbrida (SWA + Mish) com um modelo VGG-16 no CIFAR-10.