Projeto Demonstrativo 5: Reconhecimento de cenas

Raphael Soares Ramos
raphael.soares.1996@gmail.com

Departamento de Ciência da Computação
Universidade de Brasília
Campus Darcy Ribeiro, Asa Norte
Brasília-DF, CEP 70910-900, Brazil,

Abstract

Temos como vantagem do uso de aprendizado profundo a desnecessidade de engenharia de características - a própria rede o faz. Como contrapartida, necessita-se de uma grande quantidade de exemplos de treinamento. Neste trabalho isso foi mitigado pelo uso de *data augmentation*, e transferência de aprendizado ao usar pesos pré-treinados para uma maior e mais desafiadora tarefa de classificação de imagens em 1000 classes: a *ImageNet*. Os modelos utilizados neste trabalho para comparação e avaliação dos hiper-parâmetros foram: *InceptionV3*, *InceptionResNetV2* e *XceptionV3*. A melhor acurácia obtida foi de 93.43% usando a arquitetura *InceptionResNetV2*. Os hiper-parâmetros investigados foram: *dropout*, *learning rate*, *batch size* e número de épocas.

1 Introdução

O objetivo deste projeto é realizar reconhecimento de imagens. O dataset [12] abordado usa a base de imagens *15 scenes dataset*. São 15 categorias de ambientes (cenas) e um total de 1500 imagens de treino e 2985 de teste.

Redes neurais profundas são a base dos resultados do estado da arte para reconhecimento de imagens [14], detecção de objetos [6], reconstrução tridimensional de objetos [4], reconhecimento de faces [21], reconhecimento de discurso [7], *machine translation* [17], geração de legendas de imagens [22], tecnologia de carros autônomos [9], entre outros. Entretanto, treinar uma rede neural profunda é uma problema de otimização global difícil. Por isso, para o presente trabalho foi utilizado o método de *machine learning* conhecido como *transfer learning* [13]. *Transfer learning* é um metodo onde um modelo desenvolvido para uma tarefa é reusado como ponto de partida para um modelo em outra tarefa. Esse método foi utilizado neste projeto pois ele permite progresso rápido e performance melhorada para modelar a tarefa requerida.

2 Metodologia

2.1 InceptionV3

A *InceptionV3* [19] é uma rede neural convolucionária que faz convoluções fatorizadas e regularizações mais agressivas. Ela foi escolhida devido a combinação de poucos parâmetros (cerca de 23 milhões), regularização adicional com classificadores auxiliares *batch-normalized* e *label-smoothing* que permite treinar redes de alta qualidade com conjuntos de treinamento modestos.

Treinar redes neurais é complicado pelo fato que a distribuição de cada camada de entrada altera durante o treinamento assim que os parâmetros das camadas anteriores mudam. Isso desacelera o treinamento devido a necessidade de *learning rates* menores e inicialização de parâmetros mais cuidadosa. A *InceptionV3* utiliza da *batch normalization* para tentar resolver esse problema que torna muito difícil treinar modelos com saturações não lineares. Esse problema é conhecido como *internal covariate shift*. O ponto positivo da *batch-normalization* [10] é que podemos usar *learning rates* mais altas, porque a *batch-normalization* garante que não haverá ativação que será muito alta ou muito baixa. Além disso, ela possui um efeito de regularização que reduz *overfitting*. Para aumentar a estabilidade da rede neural, *batch normalization* normaliza a saída da camada anterior subtraindo a pela média do *batch* e dividindo pelo desvio padrão do *batch*.

As redes *Inception* são totalmente convolucionárias e seguem alguns princípios básicos que tentam melhorar a arquitetura da rede, como o balanceamento da largura e da profundidade da rede. Performances ótimas da rede podem ser atingidas balanceando o número de filtros por estágio e a profundidade da rede. Aumentar a largura e a profundidade contribuem para melhor qualidade da rede.

A *InceptionV3* fatora convoluções maiores em menores, visto que uma convolução com kernel 5x5 com n filtros sobre um grid com m filtros é 25/9 = 2.78 vezes computacionalmente mais caro do que uma convolução 3x3 com o mesmo número de filtros. Além disso, a *InceptionV3* também utiliza de fatoração espacial em convoluções assimétricas. Por exemplo, usar uma convolução 3x1 seguida por uma 1x3 é equivalente a "deslizar" uma rede com duas camadas com o mesmo campo receptivo como em uma convolução 3x3. Entretanto, a solução com duas camadas é 33% mais barata para o mesmo número de filtros de saída - se o número de filtros de entrada e saída são iguais. Os bancos de filtros no módulo foram expandidos (mais largos em vez de mais profundos) para remover o gargalo representacional. Se o módulo fosse mais profundo, haveria muitas reduções de dimensões, e consequentemente perda de informação.

[19] argumenta que classificadores auxiliares (introduzido por [18]) agem como regularizadores. Esse argumento é suportado pelo fato que o classificador principal da rede se sai melhor (a *InceptionV3* conseguiu um ganho absoluto de 0.4% na acurácia top-1 com classificadores auxiliares no topo da última camada 17x17) se o ramo lateral é *"batch-normalized"* ou se tem uma camada de dropout [16].

Todas essas mudanças, com exceção de *batch-normalization*, foram incorporadas já na *Inception V2*. A *InceptionV3* incorporou também: otimizador *RMSProp*; convoluções fatoradas 7x7; e *label smoothing* (um tipo de componente regularizador incorporado na *loss formula* que previne a rede de se tornar muito confiante sobre uma classe e previne *overfitting*).

2.2 Inception ResNetV2

Conexões residuais foram introduzidas por [8], onde foi apresentado evidências teóricas e práticas para as vantagens de se usar mistura aditiva de sinais para reconhecimento de imagens e detecção de objetos. Em [20] foi dada evidências empíricas de que combinar o treinamento com conexões residuais acelera o treinamento de redes *Inception* significativamente, além de se sair melhor do que redes completamente *Inception*, de mesmo custo computacional(*InceptionV4*), por uma pequena margem.

Para a versão residual das redes *Inception* foi utilizado blocos *Inception* mais baratos que o modelo original apresentado na subseção anterior 2.1. Cada bloco *Inception* é seguido por uma camada de expansão de filtro (convolução 1x1 sem ativação) que é usada para aumentar a dimensionalidade do banco de filtros antes da adição, para corresponder com a profundidade da entrada. Isso é necessário para compensar a redução na dimensionalidade induzida pelo bloco *Inception*. Além disso, foi usado *batch-normalization* apenas no topo das camadas tradicionais, mas não no topo dos somatórios. Isso foi feito para aumentar o número total de blocos *Inception*. O diagrama da rede é mostrado na **figura** 1.

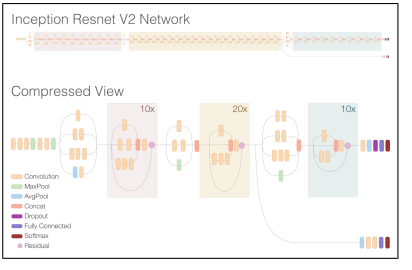


Figure 1: Diagrama comprimido da Inception ResNetV2.

Para as conexões residuais funcionarem, a entrada e saída após a convolução possui as mesmas dimensões. Logo, é usado 1x1 convoluções após a convolução original para corresponder ao tamanho das profundidades. A operação de pooling dentro dos principais módulos inception foram substituídos em favor das conexões residuais. Entretanto, essas operações continuaram nos blocos de redução. Ademais, os autores escalaram as ativações residuais por um valor entre 0.1 e 0.3, com o objetivo de aumentar a estabilidade.

2.3 Xception

A *Xception* [3] é uma nova arquitetura de rede neural convolucionária profunda inspirada na *Inception*, onde os módulos *Inception* foram substituídos por convoluções separáveis *depthwise* modificadas. A *Xception* conseguiu resultados melhores no dataset *ImageNet* [5] e até mesmo em datasets maiores com mais classes para classificação. O mais interessante é que a *Xception* tem o mesmo número de parâmetros da *InceptionV3*, o que demonstra que houve um uso mais eficientes dos parâmetros em vez de aumento da capacidade do modelo.

Na *Xception* houve uma modificação na camada *Depthwise Separable Convolution*, conforme ilustra a figura 2. Nesta camada existe uma *pointwise convolution* seguida por uma *depthwise convolution* (ordem contrária no modelo *InceptionV3*). *Depthwise convolution* é a convolução espacial canal a canal. A convolução separável em profundidade(*Depthwise convolution*) baseia-se em fatorizar a operação de convolução em duas camadas: uma convolução em profundidade que aplica um filtro para cada canal da entrada; e uma convolução

por pontos (pointwise), de tamanho 1x1, responsável por criar novas características por combinações lineares dos canais de entrada e alteram a dimensão. Como resultado, são mais baratas computacionalmente sem perdas de performance significativas em relação às convoluções completas, conforme dito anteriormente na seção 2.1.

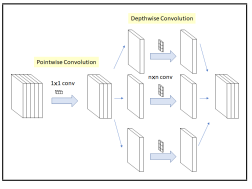


Figure 2: A convolução separável em profundidade modificada usada como um módulo *Inception* na arquitetura *Xception*. Também chamada de "versão extrema" do módulo *Inception*.

Essa modificação é motivada pelo módulo inception na *InceptionV3* onde a convolução 1x1 é feita primeira antes de qualquer *nxn* convoluções espaciais. Além disso, no módulo *Inception* original existe não linearidade após a primeira operação. No *Xception*, a convolução separável em profundidade modificada, não existe a não linearidade da função de ativação *ReLU*. Os resultados reportados pelos autores em [3] mostram que a ausência da não-linearidade leva a convergência mais rápida e performance final melhorada. Isso é um resultado interessante, visto que os autores da *InceptionV3* reportaram o oposto para módulos *Inception* em [19].

Quanto a configuração de regularização, a *InceptionV3* usa uma taxa *weight decay* (regularização L2) 4 vezes maior do que a usada na *Xception*. O Dropout utilizado por ambas as arquiteturas foi de 50% para o dataset *ImageNet*. Além disso, a arquitetura *InceptionV3* usa um mecanismo de torre auxiliar de perda/custo que serve como um mecanismo adicional de regularização, fazendo a *back-propagation* da *classification loss* mais cedo na rede. Esse mecanismo não foi utilizado na *Xception*. Todos as camadas de convolução e de convolução separável são seguidas por *batch normalization* na *Xception*.

3 Experimentos e Análise dos Resultados

Nesta seção são apresentados e analisados alguns resultados e hiper-parâmetros para todas as arquiteturas. O tamanho do conjunto de validação utilizada para todos os modelos foi de 20% do conjunto de treino fornecido (1500 imagens). Foi utilizado a função *ImageDataGenerator* [2] para o uso de *data augmentation* com o objetivo de melhorar os resultados, considerando que o tamanho do *dataset* é pequeno. 20% das imagens de treino foram utilizadas como validação com o objetivo de obter melhores resultados. O otimizador escolhido para todos os experimentos foi o [1].

3.1 InceptionV3

Avaliou-se o modelo em 3 cenários diferentes, conforme mostra a **tabela** 1. Na versão 1.0 é possível observar pelos gráficos da **figura** 3 que o *decay* utilizado de atualizar a learning rate da *learning rate* escolhida foi muito alto. Nota-se que a *loss* não convergiu, ou seja, não

chegou ao mínimo. É sabido que taxas de aprendizagem baixas precisam de mais épocas para que a função custo chegue ao mínimo e acurácia atinja o máximo.

A curva de loss/accuracy para a versão 2.0 prova que a decisão de retirar o *decay* da *learning rate* foi boa. Entretanto, é possível notar uma oscilação na *loss*. Isto provavelmente se deve ao aumento em 20% no dropout [16] utilizado. O *dropout* previne unidades de se co-adaptarem muito dropando-as junto com suas conexões, o que previne overfitting pois essas co-adaptações das unidades para diminuir a *loss* são complexas e podem não generalizar bem para dados não vistos. Contudo, o *dropout* pode aumentar o erro no início do treinamento apesar do erro/*loss* final após convergência da função de custo normalmente ser menor.

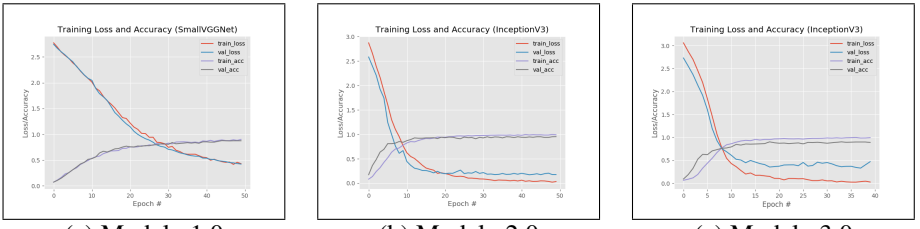
É possível notar pelo gráfico da **figura** 3 que a versão 3.0 foi a que apresentou mais sinais de *overfitting*. Há uma diferença considerável entre a *loss* na validação e no treino. Também é possível notar que a diferença entre a acurácia na validação e no treino é a maior de todos os cenários testados. Porém há dois fatores importantes a serem considerados aqui. Primeiro, o *batch size* utilizado é maior. Tamanhos de *batches* baixos podem não representar bem o conjunto de dados como um todo e podem levar o modelo a perder capacidade de generalização (considerando também o fato de que com um *batch size* menor há mais iterações). Todavia, tamanhos de *batches* altos podem diminuir a capacidade de generalização [10], visto que eles não fornecerão o verdadeiro gradiente e tendem a levar a mínimos que são muito sensíveis a perturbação dos parâmetros (*sharp minima*). Ademais, como a quantidade de imagens para treino utilizada é de 1200 imagens desconsiderando as imagens geradas usando *data augmentation*, um *batch size* de 128 já representa cerca de 10% do total de imagens, o que pode afetar a capacidade de generalização e causar um pouco de sobreajuste[1] no conjunto de treino. Segundo, foi usado uma política diferente para atualização da *learning rate*: a *learning rate* cíclica [15]. Como tamanhos de batches altos podem levar a mínimos da função de custo que são sensíveis a pertubação de parâmetros, essa política diferente de learning rate pode ser a causa da diferença observada entre o erro do treino e da validação no modelo 3.0.

Para o modelo 1.0 foi utilizado uma taxa de atualização de 0.001/50 para a *learning rate*, que é atualizada em todas as épocas. Para o modelo 2.0 foi utilizado um *callback* de reduzir pela metade a *learning rate* caso a *loss* não diminua em 5 épocas. Além disso, o treinamento da rede foi interrompido na época 45 pois não houve melhora na acurácia em 20 épocas, ou seja, a acurácia atingida na época 45 não foi maior do que a maior acurácia atingida no intervalo de épocas 20 até 45.

Table 1: Resultados da InceptionV3 em cada cenário. O modelo 1.0 obteve 0.87 de *F1 Score*, o modelo 2.0, 0.92 e o 3.0, 0.90.

	Dropout	**Learning Rate**	**Batch Size**	**Epochs**	**Acurácia**
InceptionV3-1.0	50%	0.001 Decay	32	50	86.97%
InceptionV3-2.0	70%	0.001 ReduceLR	32	45	92.43%
InceptionV3-3.0	70%	0.001 CyclicLR(triang)	64	40	89.65%

[1] vale notar que um *batch size* maior há menos atualizações nos pesos (redução da variância nas atualizações do gradiente) e pode levar a uma melhor qualidade do modelo também, já que não alterará muito os pesos pré-treinados originais



(a) Modelo 1.0 (b) Modelo 2.0 (c) Modelo 3.0

Figure 3: Resultados da InceptionV3.

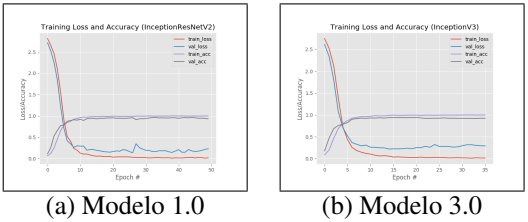
3.2 Inception ResNetV2

A rede *Inception ResNetV2* foi testada também em 3 cenários diferentes, conforme mostra a **tabela** 2. Para todos os modelos foi utilizado taxa de aprendizagem cíclica e um *early stopping para* a *loss* de 15 ou 20 épocas. Os modelos 2.0 e 3.0 foram treinados com o número máximo de 40 épocas e houve um *early stopping* na época 35 e 36, respectivamente. A decisão de diminuir o número de épocas veio após observação de que a *loss* estava convergindo para o mínimo - e a acurácia para o máximo - mais rápido do que o esperado.

Com os experimentos nessa arquitetura fortaleceu-se o argumento, apresentado na subseção anterior e por [10], de que *batches* grandes para treinamento podem diminuir a qualidade do modelo, visto que o *batch size* é a única diferença de parâmetros entre as redes 1.0 e 2.0. A rede 1.0 4 apresentou menos sinais de *overfitting*, o que já era esperado devido ao uso de maior taxa de *dropout*. Contudo, há uma maior "instabilidade" nas curvas que pode ser explicado pelo uso do modo padrão *"triangular"* [11] da política de *learning rate* cíclica. Este modo altera a *learning rate* de forma mais brusca (que provavelmente são os picos na *loss* da validação e consequentemente na acurácia dela), enquanto o modo usado na rede 3.0 *"exp_range"* altera a learning rate de forma mais suave.

Table 2: Resultados da InceptionResNetV2 em cada cenário. Todos os modelos obtiveram 0.93 de *F1 Score*, com exceção do 2.0 que obteve 0.92.

	Dropout	**Learning Rate**	**Batch Size**	**Epochs**	**Acurácia**
InceResNetV2-1.0	50%	0.001 CyclicLR(triang)	64	50	93.07%
InceResNetV2-2.0	50%	0.001 CyclicLR(exp)	128	35	92.36%
InceResNetV2-3.0	40%	0.001 CyclicLR(exp)	64	40	93.43%



(a) Modelo 1.0 (b) Modelo 3.0

Figure 4: Resultados da Inception ResNetV2.

3.3 Xception

A rede *Xception* foi testada em 3 cenários. O número de épocas máximo foi reduzido, devido a observações de rápida convergência nos modelos anteriores. Nota-se pela **figura** 5 que houve pouca diferença entre a acurácia e loss no treino e a validação. Possivelmente mais épocas poderiam resultar em uma acurácia superior à 93.43% obtida pelo modelo da seção anterior, considerando a estabilidade das curvas de aprendizagem dos modelos.

Table 3: Resultados da Xception em cada cenário. Todos os modelos *Xception* testados obtiveram 0.93 de *F1 Score*.

	Dropout	**Learning Rate**	**Batch Size**	**Epochs**	**Acurácia**
Xception-1.0	50%	0.001 CyclicLR(exp)	64	50	93.07%
Xception-2.0	70%	0.001 CyclicLR(exp)	48	25	92.66%
Xception-3.0	60%	0.001 CyclicLR(trian2)	64	25	93.23%

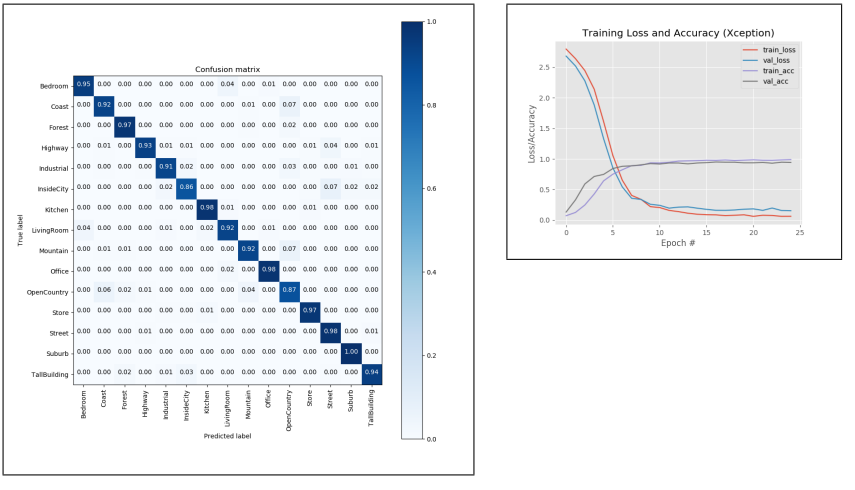


Figure 5: Matriz de confusão do modelo que obteve o melhor resultado neste projeto e curva de aprendizagem da Xception 3.0.

4 Conclusão

Como estudo futuro pode-se investigar o uso de outros otimizadores, considerando que o otimizador que forneceu os melhores resultados para os autores dos três modelos foi o *RMSprop*. A *Inception ResNetV2* se sobressaiu sobre os outros modelos, entretanto se fosse realizado mais testes acredita-se que a *Xception* conseguiria se sobressair sobre as suas similares, mesmo com um custo computacional menor ou igual. Com este trabalho foi possível notar que o hiper-parâmetro de maior importância e maior impacto para os modelos foi a taxa de aprendizagem. Não só o valor inicial da taxa, mas também a política de atualização dela.

References

[1] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[2] François Chollet et al. Keras. https://keras.io/applications/, 2015.

[3] François Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint*, pages 1610–02357, 2017.

[4] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2016.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[6] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[7] Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *International Conference on Machine Learning*, pages 1764–1772, 2014.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[9] Brody Huval, Tao Wang, Sameep Tandon, Jeff Kiske, Will Song, Joel Pazhayampallil, Mykhaylo Andriluka, Pranav Rajpurkar, Toki Migimatsu, Royce Cheng-Yue, et al. An empirical evaluation of deep learning on highway driving. *arXiv preprint arXiv:1504.01716*, 2015.

[10] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[11] Brad Kenstler. Cyclical learning rate. https://github.com/bckenstler/CLR, 2018.

[12] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Computer Vision and Pattern Recognition (CVPR)*, pages 2169–2178. IEEE, 2006.

[13] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22:1345–1359, 2010.

[14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[15] Leslie N Smith. Cyclical learning rates for training neural networks. In *Applications of Computer Vision (WACV), 2017 IEEE Winter Conference on*, pages 464–472. IEEE, 2017.

[16] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[17] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[18] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[19] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[20] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, page 12, 2017.

[21] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708, 2014.

[22] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164, 2015.