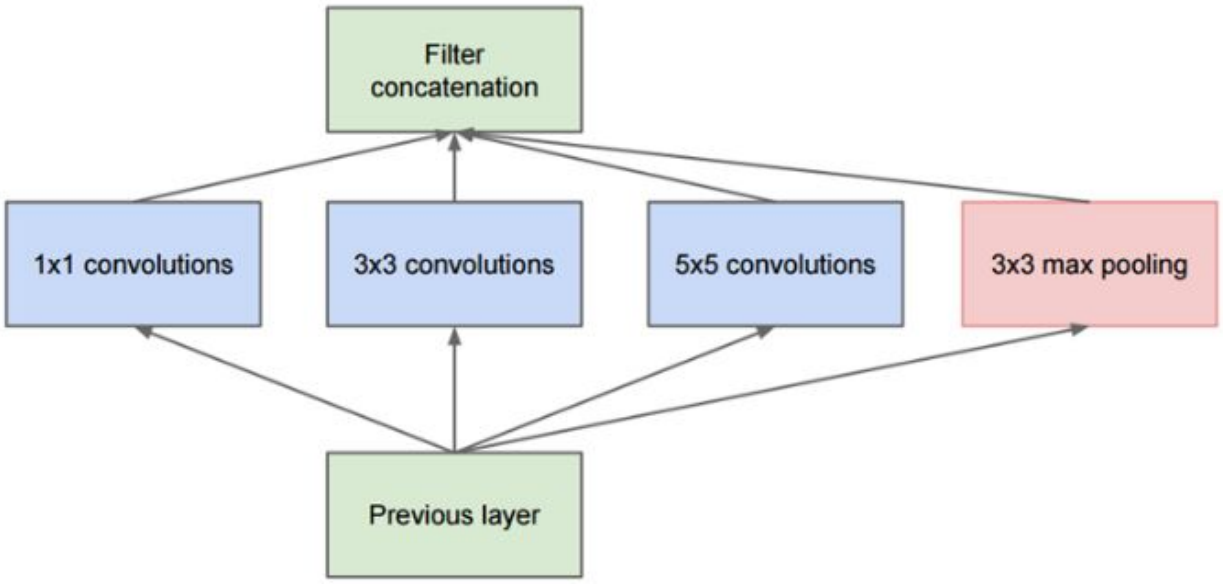
1. Apresente um exemplo de módulo inception com filtro 1x1 explicando como tal componente reduz o esforço computacional dos filtros convolucionais imediatamente seguintes apresentando as dimensões dos dados na entrada e saída de cada componente.

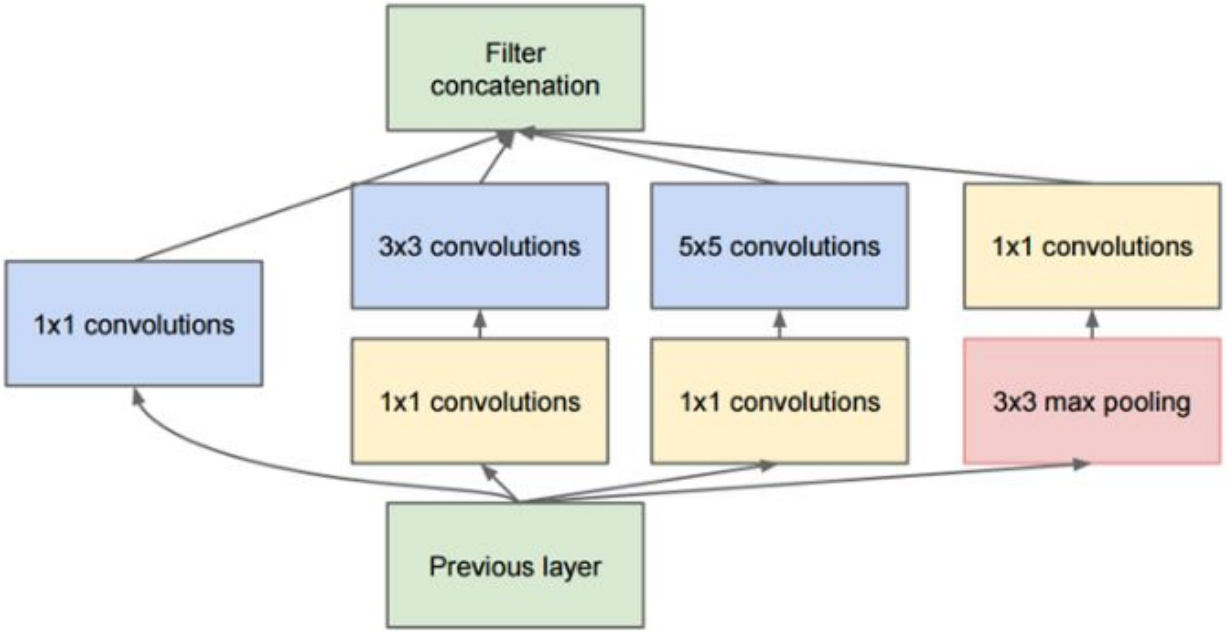
Supondo que se tenha uma dimensão de dados de entrada de 100x100x64, e que se queira produzir saídas com dimensão 128 (isto é, dobrar a quantidade de feature maps), a aplicação desta entrada nos módulos abaixo seria da forma a seguir:

Exemplo de módulo inception **sem** filtros 1x1 antes da aplicação das convoluções:



	Entrada	Saída
1x1	100x100x64	100x100x128
3x3	100x100x64	98x98x128
5x5	100x100x64	96x96x128

Exemplo de módulo inception **com** filtros 1x1 antes da aplicação das convoluções:



Aplicando-se 30 filtros convolucionais 1x1 antes de cada convolução, tem-se:

	Entrada 1x1	Saída 1x1 / Entrada Seguinte	Saída
1x1	100x100x64	-	100x100x128
3x3	100x100x64	100x100x30	98x98x128
5x5	100x100x64	100x100x30	96x96x128

A aplicação dos filtros 1x1 se explica pelo fato de que esta convolução tem o efeito de uma redução de dimensionalidade, à medida em que uma convolução 1x1 mantém a dimensão da entrada, mas modificando a quantidade de mapas resultantes, de acordo com a quantidade de filtros utilizados. Com isso, antes de se aplicar uma convolução 3x3 ou uma 5x5, a aplicação das convoluções 1x1 reduz a quantidade de dados de entrada destas convoluções, à medida em que permitem usar menos mapas do que a quantidade dos originais, que foram fornecidos ao módulo de inception. Consequentemente, isso reduz a quantidade de pesos a serem calculados no treinamento da rede, assim como a quantidade de operações a serem realizadas, reduzindo o consumo de energia, memória e o tempo de treinamento, sem perder informações ao ponto de prejudicar a performance de classificação da rede. O fato de aplicar as convoluções 1x1 também implica no aumento da não-linearidade do modelo, proporcionado pelo uso das unidades ReLU nestas convoluções.

2. Se classificadores auxiliares são posicionados nas saídas dos módulos inception, tais classificadores fazem uso de abstrações de menor nível dos dados de entrada. O desempenho de tais classificadores das camadas intermediárias tendem a ser melhores ou piores do que o classificador no topo da rede? Justifique sua resposta.

Não há como afirmar com certeza se o desempenho dos classificadores auxiliares são melhores ou piores do que o classificador final. A idéia de se usar estes classificadores intermediários não é, exatamente, para se realizar classificações intermediárias, que possam vir a ter um melhor desempenho do que a classificação do final da rede. O principal motivador do uso destes classificadores foi o fato de que uma rede tão profunda poderia sofrer o problema de vanishing gradient, visto que poderia ser difícil propagar o gradiente pelas camadas. O uso destes classificadores foi pensado pois foi considerado que isto traria ao modelo como um todo uma capacidade discriminatória maior nas camadas intermediárias, e isso ajudaria a combater o problema de vanishing gradient, à medida em que provê, também, regularização. Isto se dá pois o erro destes classificadores também é calculado e contabilizado no erro total da rede, garantindo que o treinamento destas camadas mais distantes também sejam treinadas.

3. Por qual razão o simples empilhamento de camadas convolucionais não aumenta o desempenho da rede neural? Qual a estratégia do módulo residual para contornar esse problema?

A razão pela qual o empilhamento de novas camadas convolucionais não aumenta o desempenho da rede é pelo efeito degradativo que foi verificado, à medida em que novas camadas vão sendo adicionadas em um modelo com certa profundidade. Esta degradação indica que o desempenho da rede, em uma determinada profundidade, satura, e que o erro dela passa a aumentar. Visto que este erro que aumenta também contempla um aumento no erro de treinamento, pode-se entender que este não é um erro verificado pelo fenômeno de *overfitting*, já que, neste cenário, o erro de treinamento é reduzido à medida em que o de teste aumenta.
A estratégia do módulo residual é transportar ao final de um bloco de camadas de convolução a própria entrada do bloco, x, como sendo a identidade da entrada. A idéia é que a otimização dos pesos, tendo acesso ao próprio valor da entrada, fica mais fácil de ser calculada, pois pode-se otimizar somente o mapeamento residual (a diferença entre a entrada e a transformação da convolução), ao invés do mapeamento original, sem referências.

4. O que é Batch Normalization (BN) e como esse tipo de técnica pode ser útil no contexto de redes neurais profundas?

Batch Normalization é uma técnica utilizada para combater o fenômeno de *Internal Covariate Shift*, que se refere à mudança na distribuição dos dados de entrada de uma rede. No caso de redes neurais profundas, a entrada de cada camada é afetada pelos parâmetros de todas as camadas de entrada. Isso significa que pequenas mudanças na rede são amplificadas por toda

a rede, o que causa uma mudança na distribuição da entrada em camadas internas da rede profunda, o que dificulta o treinamento, deixando-o mais lento pela necessidade de usar taxas de aprendizado reduzidas, bem como uma inicialização mais cuidadosa dos parâmetros.
O Batch Normalization atua contra estes problemas fazendo uma transformação nos dados produzidos por uma camada, antes que seja aplicada não-linearidade a eles, e esta transformação se dá somando uma média dos dados (parâmetro β) e uma multiplicação pela variância (parâmetro γ). Fazendo isso, os efeitos do *internal covariate shift* são reduzidos, e isso permite usar maiores taxas de aprendizado e reduzir o uso de dropout, além de funcionar como um regularizador do modelo.

5. Dado uma rede neural profunda convolucional qualquer, em qual ou quais camadas você adicionaria BN? Justifique sua resposta.

Em uma rede convolucional, batch normalization seria acrescentado nas camadas convolucionais, antes da aplicação dos ReLUs, pois são as camadas em que seria mais relevante promover uma distribuição mais estável dos dados de entrada.

6. Descreva pelo menos duas aplicações em que você usaria a técnica de transfer learning fine-tuning (treinamento somento da camada fully connected).

Transfer learning poderia ser aplicado em cenários em que se tem modelos pré-treinados em mãos, mas deseja-se realizar uma tarefa semelhante, com um domínio diferente, ou até mesmo um domínio semelhante, mas com uma menor disponibilidade de dados. Por exemplo, para classificação de imagens médicas que visam o reconhecimento de cânceres, dispõe-se de uma pequena quantidade de dados, dado a natureza do problema. Seria possível usar um modelo pré-treinado e retreinar somente o classificador para reconhecer imagens disponíveis. O mesmo poderia ser feito se quiséssemos usar um tradutor que traduza de inglês para português, por exemplo, e quiser retreiná-lo para ser aplicado de inglês para espanhol, por exemplo.

Exercícios práticos

1. O notebook "Traffic Sign Classification.ipynb" disponibilizado na pasta da disciplina contém uma arquitetura de rede neural com 6 camadas convolucionais para a tarefa de reconhecimento de placas de trânsito (um dos componentes importantes da tecnologia de carro autônomos). Realize alterações a partir do código disponível de modo a aprimorar a taxa de acerto atual de 98.29%. Lembre-se que 0,01% pode fazer com que o carro autônomo bata no seu carro futuramente.

Conjunto de dados: http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset

Dicas de possíveis aprimoramentos:

• Arquiteturas diferentes

• Uso de normalizações

• Alterações nas parametrizações (inicialização, Data Augmentation, normalização das imagens de entrada, taxas de aprendizados, etc)

• Ensemble de modelos

2. Fazendo uso do data set "dogs versus cat" https://www.kaggle.com/c/dogs-vs-cats/data faça um estudo comparativo das arquiteturas das redes apresentadas até o momento utilizando o binário de redes treinadas previamente fazendo uso somente de transfer learning. O código base está no arquivo "transfer_learning_cachorro_versus_gato.py"

Accuracy VGG16 = 0.91625
Accuracy VGG19 = 0.8975
Accuracy Xception = 0.98625
Accuracy Resnet = 0.96625