SLIDE 1: LEARNING STRIDES IN CONVOLUTIONAL NEURAL NETWORKS (SOFIA)

(Strides, the integer factor of the downsampling convolutional layers, are a critical hyperparameter. Are not differentiable, and would need cross-validation or discrete optimization (prohibitive in large state spaces).
So, the paper introduced Diffstride, the first downsampling layer that learns its strides jointly with the rest of the network.)

SLIDE 2: PROBLEM STATEMENT (with Spatial pooling) (SOFIA)

Convolutional layers summarize the features available in the data in terms of feature maps. But a limitation for these feature maps is that they are very sensitive regarding the location of the data (record the precise position of the features in the input; meaning that small movements in the position of the feature in the input image, will result in a different feature map).
A common approach to address this problem and make the model more robust to variations in the position of the features, is the downsampling: it generates a lower resolution version of an input signal that contains large or important structural elements, without fine details. Therefore the resolution of intermediate representations is reduced, also reducing the number of parameters to learn and the computational complexity of the whole architecture.
A possibility in order to achieve Downsampling is the Spatial Pooling, by changing, in the convolutional layer, the stride of the convolution across the image.
Stride controls how the filter convolves around the input volume. In the first case, the filter convolves around the input volume by shifting one unit at a time, because stride=1. By changing the strides to (2,2) has the effect of moving the filter 2 pixels right for each horizontal movement and 2 pixels down for each vertical movement of the filter. So it decreases the size of the resulting feature map by 75%.
(default value of strides = (1,1) for height and width movement)
(stride is normally set in a way so that the output volume is an integer and not a fraction)

But integer strides (integer factor of downsamplig) are a critical hyperparameter of such layer and tend to reduce resolution too quickly.
A more robust and common approach is to use a pooling layer, which is added after the convolutional layer (and is applied to the feature maps output).
And the approach proposed by the paper we studied consists in implementing two different pooling layers, used instead of the strides in the convolutional layers and that work in the frequency domain.
The first approach proposed is Fixed Spectral pooling.

SLIDE 3 : FIRST PROPOSED APPROACH → fixed spectral pooling (SOFIA)
Mathematical formulation of the problem

SLIDE 4 : FIRST PROPOSED APPROACH → fixed spectral pooling (SOFIA)

- It performs dimensionality reduction by truncating the representation in the frequency domain. The input image x is typically a multi-channel input and the same cropping is applied to all channels.
- Spectral pooling only requires integer output dimensions, therefore allowing also the use of non-integer strides and so a much more fine-grained downsizing.

- It preserves considerably more information w.r.t. other methods of pooling because, working in the spectral domain, cuts the higher frequencies which encode noise → fixed spectral pooling can be viewed also as a type of denoising.
- spectral pooling is differentiable wrt to its input, but not wrt the strides.

SLIDE 5 : SECOND PROPOSED APPROACH → learnable spectral pooling. (SOFIA)
This layer works on an input image that is first transformed in the frequency domain through the Discrete Fourier Transform.
This approach addresses the difficulty of searching stride parameters. In fact this time, the layer has some parameters to learn through backpropagation, that are the strides: so also the bounding box is learnable and it is parametrized by the shape of the input, a smoothness factor R and the strides, as expressed in the formulas, which compute the 1D masking functions, one along the horizontal axis and one along the vertical axis. Exploiting the conjugate symmetry of the fourier coefficients, we only consider positive frequencies along the horizontal axis, while we mirror the vertical mask around frequency zero.
Once these two 1D masks have been computed, the outer product of them generates the total mask. This final mask is applied to the fourier representation of the inputs.
Finally we crop the fourier coefficients where the mask is zero.
(the cropping operation is not differentiable wrt the strides. Therefore we apply a stop gradient operator to the mask before cropping)
and the cropped tensor is transformed back into the spatial domain (using the inverse DFT).
(same strides are learned for all the channels of the image, to ensure uniform spatial dimensions across channels.)

SLIDE 6 : resnet 18 (SOFIA)

1. There is a first layer which performs convolution (with kernel 7x7) + batch normalization + ReLU + max pooling operation (with kernel 3x3).
2. Then there are 2 identity blocks (continuous line).
   DIRE COME È IMPLEMENTATO DOPO
3. 4. 5. These three blocks are composed by a residual block (dotted line) followed by an identity block. DIRE COME È IMPLEMENTATO DOPO

At the end there is the average pooling with a flatten layer and the dense layer which will produce the output over 10 classes because of CIFAR10.

SLIDE 7 : Identity and Residual blocks (SOFIA)
This network introduces skip-connections that operate in parallel to the main branch.
Identity block: There are two convolutional layers with a kernel of size 3x3. The first one is followed by batch normalization and ReLU while the second only by batch normalization. At the end of the identity block there is the Add layer which sums the output of this sequence with the output of the skip connection that is actually the original input of the identity block. The identity block isn't modified with the introduction of spectral pooling or diffstride because strides are fixed to (1,1).
Residual block: We can divide it in two parts. The first one is made of a convolutional layer with a 3x3 kernel, batch normalization and ReLU as activation function. All these things are followed by another convolutional layer (kernel 3x3) and batch normalization. The second part is actually a parallel layer, the skip connection, which is composed of a convolutional layer. The outputs of the two parts are summed together with an Add layer followed by the ReLU function.

The residual block is modified when we introduce spectral pooling or diffstride and in particular after the first convolutional layer of the first part and after the convolution of the parallel layer we have the custom layer.

SLIDE 8: DATASET USATI e preprocessing (FEDE)
We conducted experiments with the CIFAR-10 dataset. It consists of 60.000 coloured images of size 32x32. They belong to 10 different classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck), with 6.000 images per class. There are 50.000 training images and 5.000 validation images and 5.000 test images. (These images are divided into batches (1 for the val/test and 5 for the training) in which images were randomly selected from each class.)

SLIDE 9: PREPROCESSING (FEDE)
Datas were organized through a tf.data.Dataset and preprocessed by:

- first normalizing the image wrt the mean and the standard deviation of the dataset cifar 10, which has its own specific values.

(Normalization was performed by passing the mean and variance (computed as the square of the std) to a tf.keras.layers.Normalization.)
For test dataset only normalization was performed, while for train and validation datasets, we also have:

- the padding with zeros to the specified height and width, in order to increment a bit the size of the image
- randomly cropping
- randomly flipping from left to right

We have done also some trials using a different augmentation that exploits the mixup technique but it resulted in a lower classification accuracy. The mixup technique consists in mixing up two images in order to get a third image. The two images contribute to the resulting one with different weights (the lambda in the figure) which are randomly selected.

SLIDE 10 : ESPERIMENTI (FEDE)
In the table we reported some of the experiments that were performed with strides on height and width of (2,2).
We have done training with different batch sizes and with or without mixup. The more significant results were obtained through a batch of 128 and no mix up. It is recognizable how the fixed spectral pooling brings improvements of about 2% in the accuray. But the best improvement, of about 10% in accuracy, is obtained through the method that also learns stride's value.

For spatial and fixed spectral pooling, trainings lasted 150 epochs. For learnable strides pooling, 40 epochs were considered. Also, for method 3, early stopping was exploited: in fact it is computationally heavier, since also the value of the strides is trained. Also, from the first trainings done, we saw that after 30/40 epochs the accuracy stabilized, not increasing anymore.

After having found the best parameter setting, we have used *no mixup* + *batch of 128* to do experiments changing strides sizes: (2,2,3), (3,1,3), (3,1,2) where the i-th number represents the size of strides used in the i-th residual block if the Resnet-18.

SLIDE 11 : CONCLUSIONI (FEDE)
Therefore, using Diffstride as a drop-in replacement to standard downsampling layers outperforms them, also removing the need of cross-validating the strides value.
But, a limitation of the approach proposed is that pooling in the spectral domain comes with a higher computational cost, but improvements can be done in order to alleviate computations (for example computing the convolution in the Fourier domain ad an element-wise multiplication and summation over channels).

SLIDE 12 : Additional implementation with PHC layer (FEDE)
We also substituted the classical Conv2d layers in the network with Parametrized Hypercomplex Convolutional layers, which allow to reduce the overall number of parameters by a factor of N. The reduction factor can be chosen by the user, making PHC layers adaptable to a lot of applications and HW.

PHC exploit the properties of hypercomplex algebra that allows the parameters sharing inside the layers. In particular it generalized these hypercomplex multiplications exploited by Quaternion models as sum of Kronecker products.
Given
Aì = matrices that describe the algebra rules
Fì = i-th batch of convolutional filters that are arranged by following the algebra rules, to compose the final weight matrix.
PHC is based on a sum of Kronecker products between these 2 learnable matrices A and F.
The Kronecker product is a generalization of the vector outer product that can be parametrized by N.
A and F are learned during training and their values are used to build the definitive tensor H.
Therefore, given x as input, the PHC convolution given by Y = …., where H encapsulates the filters of the convolution, and b is the bias.

NO:
In fact PHC is able to exploit the proper algebra from input data and to capture internal correlations among the image channels, saving the 66% of parameters.
This method takes the idea of hypercomplex multiplications exploited by Quaternion models in order to design interactions between imaginary units, thus involving less parameters. But PHC generalized these hypercomplex multiplications exploited by Quaternion models as sum of Kronecker products, going beyond quaternion algebra

SLIDE 13 : Experiments (FEDE)
PHC layer used instead of 2D convolutions in the whole network
Experiments were done with:

- N = 3 → ⅓ parameters of the previous experiments (from 12.000 to 4.000)
- Strides initialization for the 3 residual layers = *(2, 2, 2)*

→ riducendo il numero di parametri allenabili nella rete abbiamo comunque raggiunto un 50% di accuracy nella classificazione di 10 classi, sempre con Dataset cifar 10. E abbiamo il PHC layer sia con la convoluzione basata su strides che su quelle che agisce nel dominio delle frequenze con stride fisse, dimostrando che anche in questo caso, agire nel dominio delle frequenze porta a miglioramenti nei valori dell'accuracy.

FEDE: REVISIONE DISCORSO

SLIDE 8: DATASET USATI e preprocessing

- We have used for the experiments the CIFAR-10 dataset.
- It consists of 60.000 coloured images of size 32x32.
- There are 10 different classes such as airplane, horse and ship; each class has 6.000 images.
- There are 50.000 training images and 5.000 validation images and 5.000 test images.
- We have used batches of 5 images for the training set and of 1 image for validation and test sets.

- Abbiamo usato per gli esperimenti il dataset CIFAR-10 che consiste di 60.000 immagini a colori della dimensione 32x32.
- Ci sono 10 classi differenti come aeroplano, cavallo, barca e ogni classe ha 6.000 immagini.
- Ci sono 50.000 immagini per il training e 5.000 immagini per il validation e 5.000 per il test set.
- Abbiamo deciso di usare batches di 5 immagini nel caso del training set e di 1 immagine per il validation set e test set.

SLIDE 9: PREPROCESSING

About the preprocessing, first there is the normalization of the image with respect to the mean and the standard deviation of the CIFAR-10 which has its own specific values.
While for the test set there is only the normalization as preprocessing, for the training set and validation set we have also applied:

- padding with zeros in order to increment a bit the size of the image
- random cropping
- random flipping from left to right

Actually we have also done some trials using another type of data augmentation which is the mixup (it consists in mixing up two images in order to get a third image; the two images contribute to the resulting one with different weights which are randomly selected).

Per il preprocessing, prima di tutto abbiamo normalizzato le immagini rispetto la media e la standard deviation di CIFAR-10 che ha dei valori già specificati.
Mentre per il test set abbiamo applicato solo la normalization, per il training set e per il validation set abbiamo applicato anche:

- padding di zeri per aumentare la dimensione dell'immagine
- random cropping
- random flipping da sinistra a destra

In realtà abbiamo fatto anche delle prove utilizzando il mixup e cioè abbiamo creato una terza immagine a partire da 2 immagini presenti nel dataset le quali contribuiscono alla realizzazione con due pesi diversi.

SLIDE 10 : ESPERIMENTI
In the table we reported some of the experiments that were performed with strides on height and width of (2,2).
We have done training using different batch sizes and with or without mixup.
The more significant result has been obtained using a batch of 128 and no mix up.

It is evident how the fixed spectral pooling brings improvements of about 2% in the accuray. but actually the best improvement, of about 10%, is obtained through DiffStride.

For spatial and fixed spectral pooling, training is made on 150 epochs.
For diffstride, 40 epochs were considered and moreover here early stopping was exploited: in fact it is computationally heavier, since also the value of the strides is trained. We have seen that after 30/40 epochs the accuracy stabilized, not increasing anymore.

Qui nella tabella abbiamo riportato alcuni esperimenti che abbiamo fatto usando le strides 2x2.
Abbiamo eseguito i training usando differenti differenti dimensioni per i batch e con e senza il mixup.
Il risultato più significativo è stato ottenuto con il batch di dimensione 128 e senza mixup.

É evidente come il fixed spectral pooling porti un miglioramento sull'accuracy del 2% ma il risultato migliore è ovviamente quello di DiffStride che aumenta l'accuracy del 10%.

Abbiamo usato 150 epoche per i primi due metodi.
Per DiffStride abbiamo usato solo 40 epoche e usato l'early stopping poiché questo metodo è più pesante a livello computazionale dato che deve imparare anche le dimensioni delle stride. Abbiamo settato le epoche a 40 perchè abbiamo visto dai primi esperimenti che dopo 40 epoche l'accuracy non cresceva più.

SLIDE 11 : ALTRI ESPERIMENTI E CONCLUSIONI
After having found the best parameter setting, we have used *no mixup + batch of 128* to do experiments changing strides sizes: (2,2,3), (3,1,3), (3,1,2) where the i-th number represents the size of strides used in the i-th residual block if the Resnet-18.

So at the end we have proved how DiffStride outperforms the standard downsampling method and also the fixed spectral pooling.

A limitation of the approach proposed is that pooling in the spectral domain comes with a higher computational cost and moreover stride dimension becomes an additional parameter that the network has to learn.

Dopo aver capito che batch a 128 e no mixup era il miglior settaggio, abbiamo fatto altri esperimenti cambiando le dimensioni delle strides (2,2,3), (3,1,3), (3,1,2) dove il numero i-esimo corrisponde alla dimensione delle stride nell'i-esimo blocco della resnet.

Quindi alla fine possiamo concludere che DiffStride sia migliore del metodo standard e anche di fixed spectral pooling.

Un aspetto negativo tuttavia è presente e cioè fare pooling nello spectral pooling ha un costo computazionale maggiore considerando anche che la rete deve imparare altri parametri e cioè la dimensione ottimale delle stride.

SLIDE 12 : IMPLEMENTAZIONE AGGIUNTIVA CON PHC LAYER
For this additional implementation we have substituted the classical Conv2d layers in the network with Parameterized Hypercomplex Convolutional layers, which allow to reduce the number of parameters by a factor of N. This reduction factor is a hyperparameter.

PHC exploits the properties of hypercomplex algebra that allows the parameters sharing inside the layers. In particular it generalizes hypercomplex multiplications as sum of Kronecker products (they are a generalization of the vector outer product that can be parametrized by N).
Given
A_i = matrices that describe the algebra rules

F_i = i-th batch of convolutional filters that are arranged to compose the final weight matrix.
PHC is based on a sum of Kronecker products between these 2 learnable matrices A and F.
A and F are learned during training and their values are used to build the definitive tensor H.
Therefore, given x as input, the PHC convolution is given by Y = …., where H encapsulates the filters of the convolution, and b is the bias.

Per questa implementazione aggiuntiva, abbiamo sostituito i convolutional layers classici con i PHC ovvero Parameterized Hypercomplex Convolutional layers che permettono di ridurre il numero dei parametri di un fattore N che è un nuovo iper-parametro della rete.

PHC si basa sulle proprietà dell'algebra iper-complessa che permette quindi una condivisione di parametri all'interno delle stesso layer. In particolare pHC generalizza le moltiplicazioni iper-complesse trattandole come una somma di prodotti di Kronecker che a loro volta sono una generalizzazione dell'outer product.

Dati
A_i = matrici che descrivono l'algebra

F_i = i-esimo batch dei filtri convoluzionali
PHC si basa quindi sulla somma di prodotti di Kronecker tra le matrici A e F, matrici che sono imparate durante il training e che vanno a costituire il tensore H.
Quindi alla fine dato l'input x, PHC è dato da Y=... dove H incapsula i filtri per la convoluzione e b è il bias.

NO:
In fact PHC is able to exploit the proper algebra from input data and to capture internal correlations among the image channels, saving the 66% of parameters.
This method takes the idea of hypercomplex multiplications exploited by Quaternion models in order to design interactions between imaginary units, thus involving less parameters. But PHC generalized these hypercomplex multiplications exploited by Quaternion models as sum of Kronecker products, going beyond quaternion algebra

SLIDE 13 : ESPERIMENTI
So to recap we have used PHC layer instead of 2D convolutions in the whole network.

Experiments were done with:

- N = 3 → ⅓ parameters of the previous experiments (from 12.000 to 4.000)
- Strides initialization for the 3 residual layers = *(2, 2, 2)*

→ despite we have reduced the number of trainable parameters, we have reached 50% of accuracy

for a classification on 10 classes using also here CIFAR-10 as dataset.
We have applied PHC both in strided convolution implementation and in fixed spectral pooling implementation and also in this case we have shown how working in the frequency domain is better from the accuracy point of view.

Quindi per riassumere, abbiamo applicato il layer PHC al posto delle convoluzioni 2D in tutta la rete. Gli esperimenti sono stati fatti con:

- N = 3 e quindi abbiamo solo ⅓ dei parametri rispetto agli altri esperimenti (da 12.000 a 4.000 parametri)
- L'inizializzazione delle stride a (2,2,2)

Nonostante abbiamo ridotto il numero dei parametri da imparare, abbiamo comunque raggiunto una accuracy del 50% per una classificazione su 10 classi usando anche qui CIFAR-10 come dataset. Abbiamo applicato PHC sia nell'implementazione con strided convolution sia per quella con fixed spectral pooling e anche in questo caso abbiamo dimostrato come lavorare nel dominio delle frequenze porti ad una accuracy superiore.