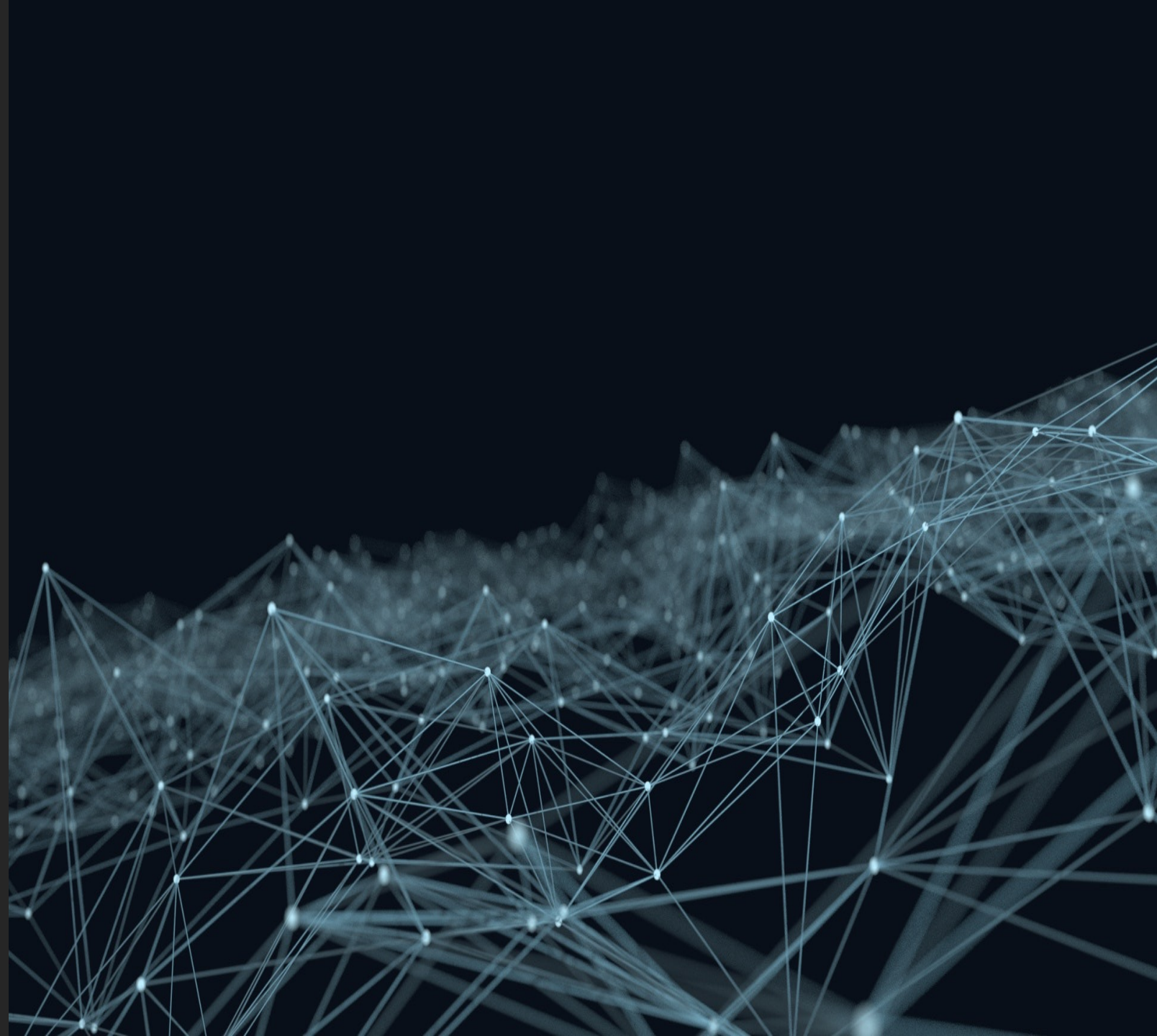
DropBlock:

A regularization method for Convolutional Networks.

Ludovico Ottobre, 1712005
Gianmarco Evangelista, 1711818

La Sapienza, University of Rome



Background

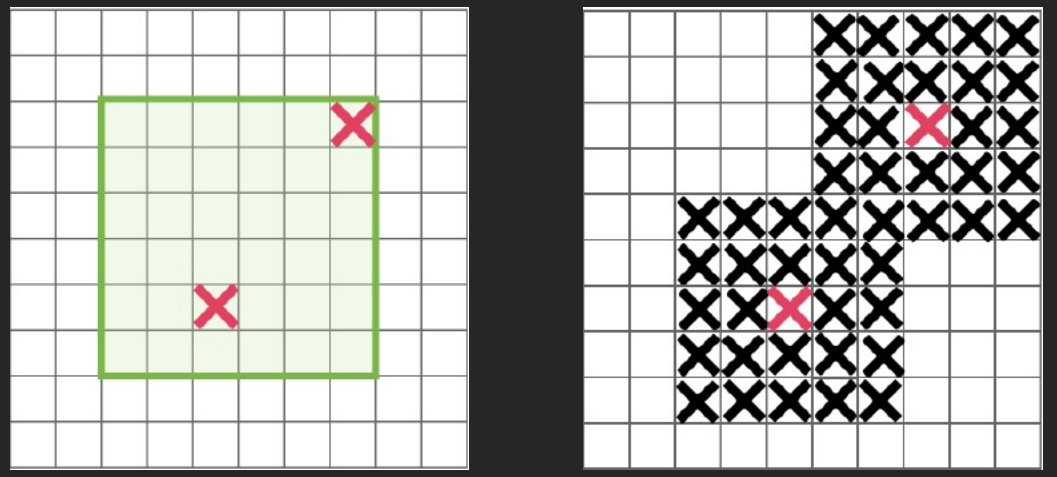
Regularization is a technique used to reduce the errors by fitting the function appropriately on the given training set and avoid overfitting. It refers to a model that models the training data too well.

Dropblock, a form of structured Dropout, improves the performances of Dropout especially in deep architectures with skip connections where skip some layer in the neural network and feed the output of one layer as the input to the next layers where units in contiguous region are dropped together.

Ludovico Ottobre, Gianmarco Evangelista.

Introduction

DropBlock method was introduced to combat the major drawback of Dropout being dropping features randomly which prove to be effective strategy for fully connected networks but less fruitful when it comes to convolutional layers wherein features are spatially correlated. DropBlock technique discards features in a contiguous correlated area called *block*. By doing so, it is able to fulfill the purpose of generating simpler model and to put in the concept of learning a fraction of the weights in the network in each training iteration to penalize the weight matrix which in turn reduces overfitting.



(a) (b)

Mask sampling in DropBlock. (a) On every feature map, similar to dropout, we first sample a mask M. (b) Every zero entry on M is expanded to *block_size X block_size* zero block.

DropBlock

The principal parameters for DropBlock algorithm are the size of the block to be dropped i.e. ***block_size*** and how many activation units to be dropped i.e. **γ** with each feature channel having its DropBlock mask.

$$\gamma = \frac{1 - keep\_prob}{blocksize^2} \frac{featsize^2}{(featsize - blocksize + 1)^2}$$

keep_prob is the threshold probability set wherein all the elements whose probability is less than keep_prob are effectively removed.
featsize is the size of feature map.
featsize - blocksize + 1 is the valid seed region.

DropBlock

Input Feature Map

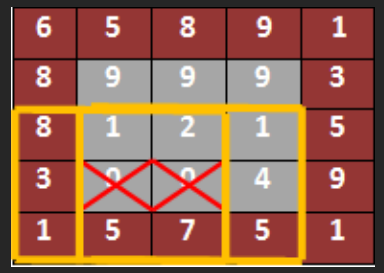
6	5	8	9	1
8	9	9	9	3
8	1	2	1	5
3	5	2	4	9
1	5	7	5	1

Mask Obtained

1	1	1
1	1	1
0	0	1

Applying the obtained mask over feature map

6	5	8	9	1
8	9X1	9X1	9X1	3
8	1X1	2X1	1X1	5
3	5X0	2X0	4X1	9
1	5	7	5	1



6	5	8	9	1
8	9	9	9	3
0	0	0	0	5
0	0	0	0	9
0	0	0	0	1

This mask is obtained by applying *Bernoulli* on *gamma* over the feature map.

Where *gamma* is obtained using the above formula keeping a fixed *block_size* and *keep_prob*.

A spatial square mask with *height* and *width* corresponding to the *block_size* is created with center being each 0 position in the mask obtained.
All the values in this mask are set to 0.

DropBlock algorithm discards features in correlated area and the contiguous region of the feature map is dropped together.

Ludovico Ottobre, Gianmarco Evangelista.

Properties of DropBlock

In our implementation, we set a constant *block_size* for all feature maps; DropBlock resembles *Dropout* when *block_size = 1* and resembles *SpatialDropout* when *block_size* covers the full feature map.

In practice, we do not explicitly set **γ**. As stated earlier, **γ** controls the number of features to drop. The binary mask will be sampled with the *Bernoulli* distribution with mean *1 − keep_prob.*

Following the idea proposed by the paper, we implemented a *callback function* that allowed us to decrease the *keep_prob* value over time. We use a linear scheme to decrease the value in question by acertain amount for each epoch.

Scheduled DropBlock

```
[ ] class myCallback(keras.callbacks.Callback):
        def on_epoch_end(self, epoch,logs={}):
            for i in range(31): #number of DropBlock layer in the model
              s = str(i)        #string in order to recognize layer
              # vary the rate wrt epochs
              self.model.get_layer(name = s).keep_prob -= 0.07
```

```
  history = model.fit(x_train, y_train,
                batch_size=batch_size,
                epochs=epochs,
                validation_data=(x_test, y_test),
                shuffle=True,
                callbacks = myCallback())
```

Pseudocode of DropBlock

Algorithm 1 DropBlock

Input: output activations of a layer (A), $block\_size$, γ, $mode$
if $mode == Inference$ **then**
    return A
end if
Randomly sample mask M: $M_{i,j} \sim Bernoulli(\gamma)$
For each zero position $M_{i,j}$, create a spatial square mask with the center being $M_{i,j}$, the width, height being $block\_size$ and set all the values of M in the square to be zero
Apply the mask: $A = A \times M$
Normalize the features: $A = A \times \textbf{count}(M)/\textbf{count\_ones}(M)$

Implementation of DropBlock

```python
from keras import backend as K


import copy

class NN_DropBlock(tf.keras.layers.Layer):

    def __init__(self, block_size, keep_prob, **kwargs):
        super(NN_DropBlock, self).__init__(**kwargs)
        self.block_size = block_size
        self.keep_prob = keep_prob

    def call(self, x, inference=None):

        # During inference, we do not Drop Blocks
        if inference == None:
            return x

        #padding
        z = self.block_size//2

        # Calculate Gamma
        feature_size = int(x.shape[-1])
        gamma = ((1-self.keep_prob)/(self.block_size**2)) * ((feature_size**2) / ((feature_size-self.block_size+1)**2))

        # Randomly sample mask
        sample_mask = tf.nn.relu(tf.sign(gamma - tf.random_uniform((feature_size-(z*2), feature_size-(z*2)), minval=0, maxval=1, dtype=tf.float32)))

        # The above code creates a matrix of zeros and samples ones from the distribution
        # We would like to flip all of these values
        sample_mask = 1-sample_mask

        # Pad the mask with ones
        sample_mask = np.pad(sample_mask, pad_width=z, mode='constant', constant_values=1)

        # For each 0, create spatial square mask of shape (block_size x block_size)
        spatial_square_mask = copy.copy(sample_mask)
        for i in range(feature_size):
            for j in range(feature_size):
                if sample_mask[i, j]==0:
                    spatial_square_mask[i-z : i+z+1, j-z : j+z+1] = 0

        spatial_square_mask = spatial_square_mask.reshape((1, feature_size, feature_size))
```

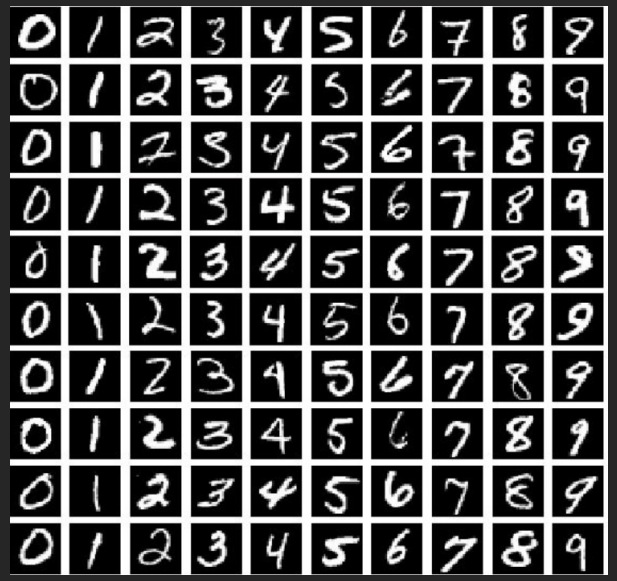
```python
        # Pad the mask with ones
        sample_mask = np.pad(sample_mask, pad_width=z, mode='constant', constant_values=1)

        # For each 0, create spatial square mask of shape (block_size x block_size)
        spatial_square_mask = copy.copy(sample_mask)
        for i in range(feature_size):
            for j in range(feature_size):
                if sample_mask[i, j]==0:
                    spatial_square_mask[i-z : i+z+1, j-z : j+z+1] = 0

        spatial_square_mask = spatial_square_mask.reshape((1, feature_size, feature_size))

        # Apply the mask
        x = x * np.repeat(spatial_square_mask, x.shape[1], 0)

        # Normalize the features
        count = np.prod(spatial_square_mask.shape)
        count_ones = np.count_nonzero(spatial_square_mask == 1)
        x = x * count / count_ones

        return x

    def get_config(self):
        config = {'block_size': self.block_size,
                  'keep_prob': self.keep_prob
                  }
        base_config = super(NN_DropBlock, self).get_config()
        return dict(list(base_config.items()) + list(config.items()))

    def compute_output_shape(self, input_shape):
        return input_shape
```


Ludovico Ottobre, Gianmarco Evangelista.

Datasets

Differently from original paper, in the project we developed we have used two of the most commonly used datasets, namely MNIST and CIFAR-10.

CIFAR-10



MNIST

Datasets details

CIFAR-10 dataset is composed by 60,000 of 32x32 colour images from 10 classes, such as airplane, automobile, bird, cat, deer, dog, frog,horse, ship, truck. We have chosen to have 50,000 images for the training datasetand 10,000 for the test set.

The MNIST database of handwritten digits, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

Convolutional networks

In order to evaluate the correctness and functionality of *DropBlock*, we decided to use three different neural networks: *custom CNN*, *DenseNet* and *ResNet*. The networks were chosen to be able to analyze and compare the results obtained with those that were already available to us.
Each of the networks was processed in five different ways: default, with *Dropout*, with *SpatialDropout*, with *DropBlock* (*block_size = 3*) and with *DropBlock (block_size = 7*).

Since the DropBlock, as suggested by the paper under analysis, is useful and efficient especially for networks with skip connection, in addition to considering ResNet, we have chosen DenseNet for our analysis. In custom CNN, as we will see later, DropBlock does not guarantee improvements over Dropout.

ResNet-50 Experiment

ResNet-50 is a CNN architecture widely used for image recognition. We have applied different regularization techniques on ResNet-50 in order to compare the result with DropBlock.

The configuration we chose to be able to insert DropBlock inside the model was to put the regularization method at the end of each *Conv - Batch – Activation* block, then immediately after the Relu activation function.

We have inserted DropBlock inside *Identity* and *Conv* blocks.

Identity block implementation

```
# First component of main path
X = Conv2D(filters = F1, kernel_size = (1, 1), strides = (1,1), padding = 'valid', name = conv_name_base + '2a', kernel_initializer = glorot_uniform(seed=0))(X)
X = BatchNormalization(axis = 3, name = bn_name_base + '2a')(X)
X = Activation('relu')(X)
X = NN_DropBlock(block_size=7, keep_prob=0.7, name = str(c[0]))(X)
c[0] += 1

# Second component of main path
X = Conv2D(filters = F2, kernel_size = (f, f), strides = (1,1), padding = 'same', name = conv_name_base + '2b', kernel_initializer = glorot_uniform(seed=0))(X)
X = BatchNormalization(axis = 3, name = bn_name_base + '2b')(X)
X = Activation('relu')(X)
X = NN_DropBlock(block_size=7, keep_prob=0.7, name = str(c[0]))(X)
c[0] += 1

# Third component of main path
X = Conv2D(filters = F3, kernel_size = (1, 1), strides = (1,1), padding = 'valid', name = conv_name_base + '2c', kernel_initializer = glorot_uniform(seed=0))(X)
X = BatchNormalization(axis = 3, name = bn_name_base + '2c')(X)

# Final step: Add shortcut value to main path, and pass it through a RELU activation
X = Add()([X,X_shortcut])
X = Activation('relu')(X)
```

Convolutional block implementation

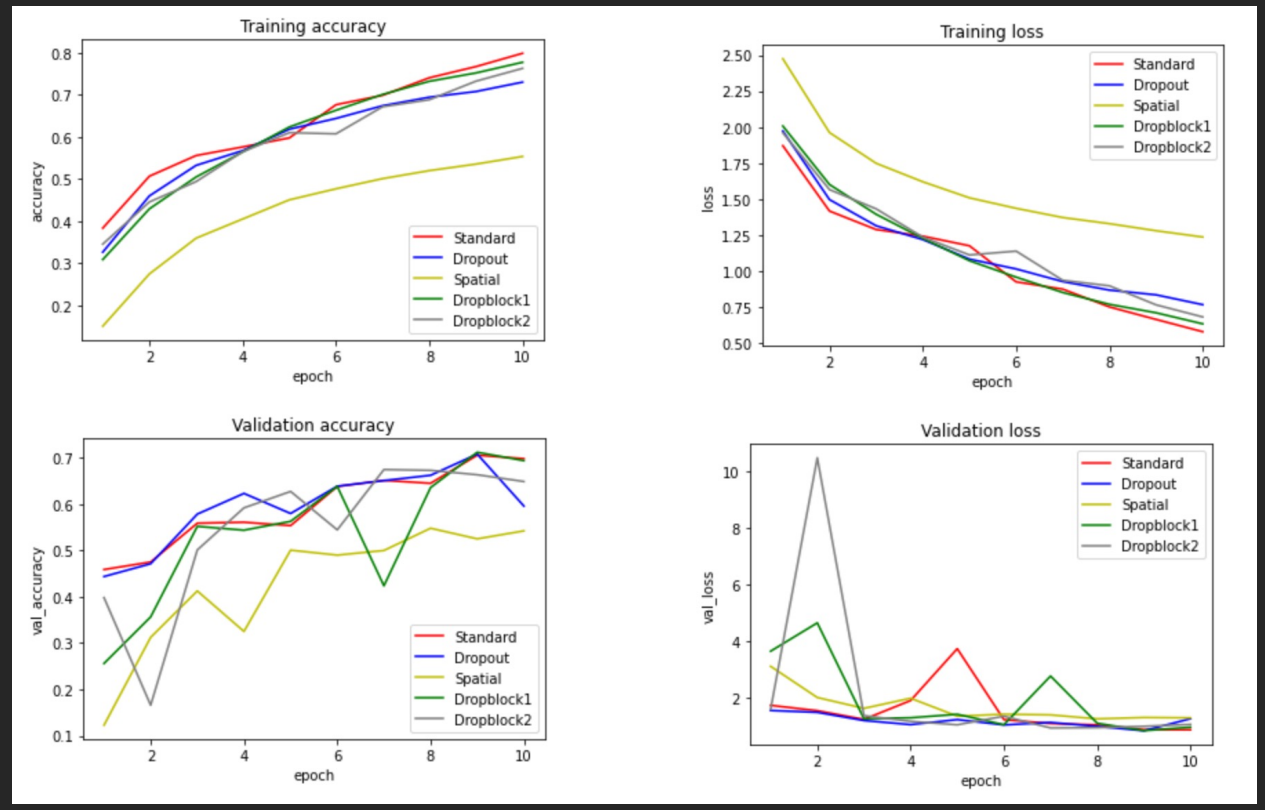
```
##### MAIN PATH #####
# First component of main path
X = Conv2D(F1, (1, 1), strides = (s,s), name = conv_name_base + '2a',padding = 'valid', kernel_initializer = glorot_uniform(seed=0))(X)
X = BatchNormalization(axis = 3, name = bn_name_base + '2a')(X)
X = Activation('relu')(X)
X = NN_DropBlock(block_size=7, keep_prob=0.7, name = str(c[0]))(X)
c[0] += 1

# Second component of main path
X = Conv2D(F2, (f, f), strides = (1,1), name = conv_name_base + '2b', padding = 'same', kernel_initializer = glorot_uniform(seed=0))(X)
X = BatchNormalization(axis = 3, name = bn_name_base + '2b')(X)
X = Activation('relu')(X)
X = NN_DropBlock(block_size=7, keep_prob=0.7, name = str(c[0]))(X)
c[0] += 1

# Third component of main path
X = Conv2D(F3, (1, 1), strides = (1,1), name = conv_name_base + '2c', padding = 'valid', kernel_initializer = glorot_uniform(seed=0))(X)
X = BatchNormalization(axis = 3, name = bn_name_base + '2c')(X)

##### SHORTCUT PATH ####
X_shortcut = Conv2D(F3, (1, 1), strides = (s,s), name = conv_name_base + '1', padding = 'valid', kernel_initializer = glorot_uniform(seed=0))(X_shortcut)
X_shortcut = BatchNormalization(axis = 3, name = bn_name_base + '1')(X_shortcut)

# Final step: Add shortcut value to main path, and pass it through a RELU activation
X = Add()([X,X_shortcut])
X = Activation('relu')(X)
```

ResNet-50 Experiment

In the following table and graphs we can see the performances on CIFAR-10.

MODEL	Accuracy CIFAR-10	Accuracy MNIST
ResNet-50	0.6878	0.9900
ResNet-50 + *Dropout(0.25)*	0.6619	0.9920
ResNet-50 + *SpatialDropout(0.7)*	0.5477	0.9807
ResNet-50 + *DropBlock(bs = 3, kp = 0.9)*	0.6935	0.9898
ResNet-50 + *DropBlock(bs = 7, kp = 0.9)*	0.6743	0.9876

ResNet-50 accuarcy



ResNet-50 performances

Ludovico Ottobre, Gianmarco Evangelista.

DenseNet Experiment

DenseNet is quite similar to ResNet with some fundamental differences. ResNet uses an additive method that merges the previous layer (identity) with the future layer, whereas *DenseNet* concatenates the output of the previous layer with the future layer.
It results to be, like ResNet, a model with *skip connection*. By using a skip connection, we provide an alternative path for the gradient (with backpropagation).

The configuration we have chosen to be able to insert DropBlock within DenseNet, follows the one that was the choice made in the case of ResNet. Again, we inserted DropBlock at the end of each *Conv - Batch – Activation* block, so right after the Relu activation function.

batch_normalization (BatchNorma	(None, 8, 8, 64)	256	max_pooling2d[0][0]
re_lu (ReLU)	(None, 8, 8, 64)	0	batch_normalization[0][0]
0 (DropBlock2D)	(None, 8, 8, 64)	128	re_lu[0][0]
conv2d_1 (Conv2D)	(None, 8, 8, 128)	8320	0[0][0]
batch_normalization_1 (BatchNor	(None, 8, 8, 128)	512	conv2d_1[0][0]
re_lu_1 (ReLU)	(None, 8, 8, 128)	0	batch_normalization_1[0][0]
1 (DropBlock2D)	(None, 8, 8, 128)	128	re_lu_1[0][0]
conv2d_2 (Conv2D)	(None, 8, 8, 32)	36896	1[0][0]
concatenate (Concatenate)	(None, 8, 8, 96)	0	max_pooling2d[0][0] conv2d_2[0][0]

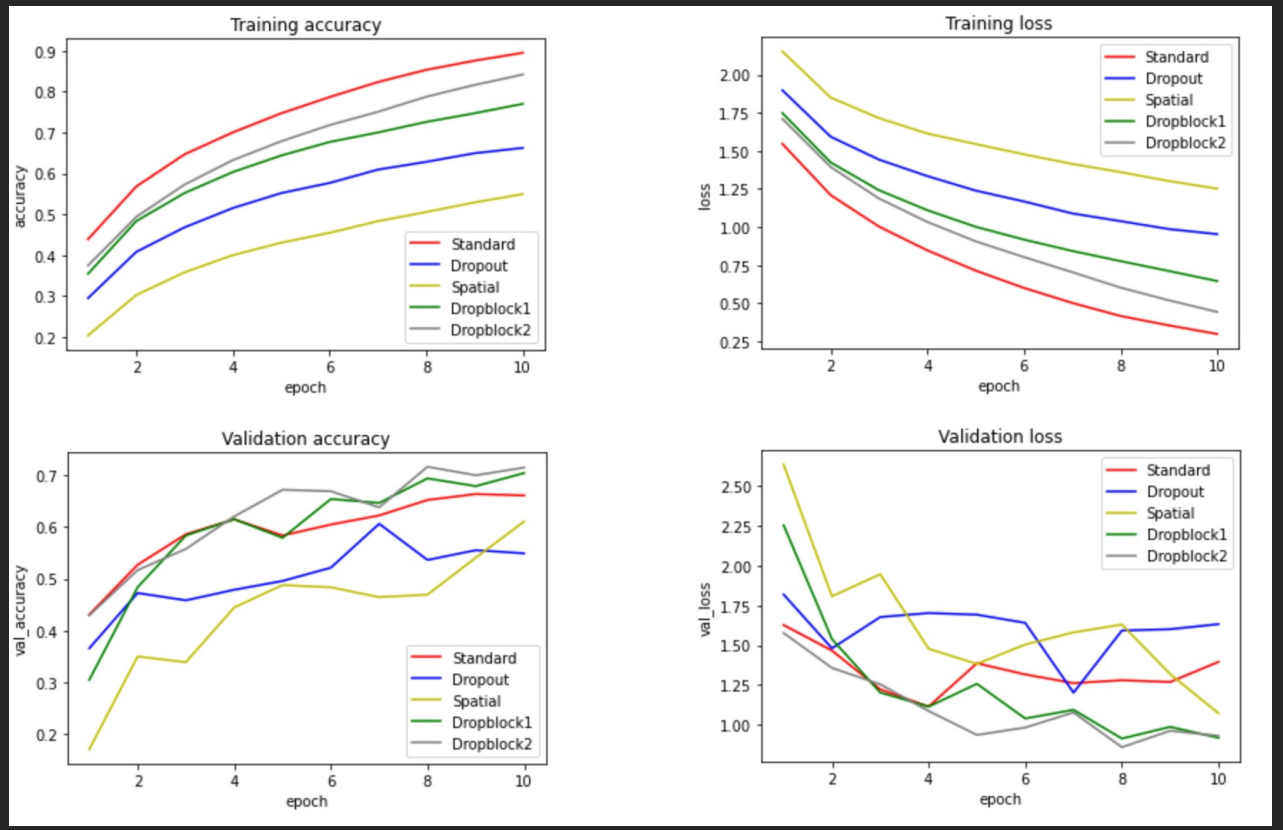
Ludovico Ottobre, Gianmarco Evangelista.

DenseNet Experiment

In the following table and graphs we can see the performances on CIFAR-10.

MODEL	Accuracy CIFAR-10	Accuracy MNIST
DenseNet	0.6581	0.9899
DenseNet + *Dropout(0.25)*	0.6060	0.9893
DenseNet + *SpatialDropout(0.3)*	0.6105	0.9799
DenseNet + *DropBlock(bs = 3, kp = 0.9)*	0.7040	0.9910
DenseNet-50 + *DropBlock(bs = 7, kp = 0.9)*	0.7146	0.9919

DenseNet accuracy



DenseNet performances

Ludovico Ottobre, Gianmarco Evangelista.

CustomNet Experiment

This CNN has be done in order to evaluate DropBlock in a different architecture no skip connection. In this model we have a lower number of parameters compared to the architectures analyzed previously.

We have inserted Dropblock after *MaxPooling2D* since it was the best position in order to obtain the better result. In this model we have a number of epochs equal to 100 so we have a more gradual decrease of keep_prob.

CustomNet

```
x_train shape: (50000, 32, 32, 3)
50000 train samples
10000 test samples
Model: "sequential_4"
```

Layer (type)	Output Shape	Param #
conv2d_24 (Conv2D)	(None, 32, 32, 32)	896
activation_24 (Activation)	(None, 32, 32, 32)	0
batch_normalization_24 (Batc	(None, 32, 32, 32)	128
conv2d_25 (Conv2D)	(None, 32, 32, 32)	9248
activation_25 (Activation)	(None, 32, 32, 32)	0
batch_normalization_25 (Batc	(None, 32, 32, 32)	128
max_pooling2d_12 (MaxPooling	(None, 16, 16, 32)	0
Drop1 (DropBlock2D)	(None, 16, 16, 32)	512
conv2d_26 (Conv2D)	(None, 16, 16, 64)	18496
activation_26 (Activation)	(None, 16, 16, 64)	0
batch_normalization_26 (Batc	(None, 16, 16, 64)	256
conv2d_27 (Conv2D)	(None, 16, 16, 64)	36928
activation_27 (Activation)	(None, 16, 16, 64)	0
batch_normalization_27 (Batc	(None, 16, 16, 64)	256
max_pooling2d_13 (MaxPooling	(None, 8, 8, 64)	0
Drop2 (DropBlock2D)	(None, 8, 8, 64)	128
conv2d_28 (Conv2D)	(None, 8, 8, 128)	73856
activation_28 (Activation)	(None, 8, 8, 128)	0
batch_normalization_28 (Batc	(None, 8, 8, 128)	512
conv2d_29 (Conv2D)	(None, 8, 8, 128)	147584
activation_29 (Activation)	(None, 8, 8, 128)	0
batch_normalization_29 (Batc	(None, 8, 8, 128)	512
max_pooling2d_14 (MaxPooling	(None, 4, 4, 128)	0
Drop3 (DropBlock2D)	(None, 4, 4, 128)	32
flatten_4 (Flatten)	(None, 2048)	0
dense_4 (Dense)	(None, 10)	20490

```
Total params: 309,962
Trainable params: 309,066
Non-trainable params: 896
```

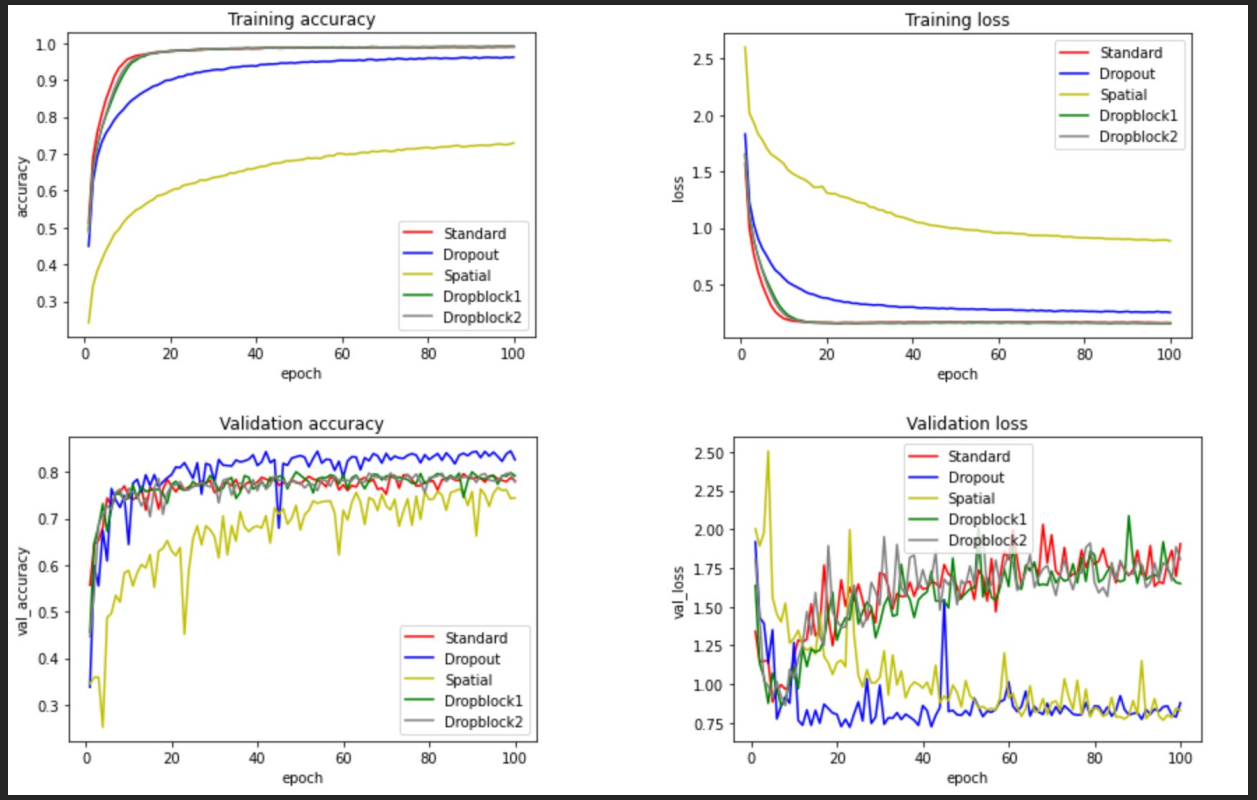

Ludovico Ottobre, Gianmarco Evangelista.

CustomNet Experiment

In the following table and graphs we can see the performances.

MODEL	Accuracy CIFAR-10	Accuracy MNIST
Custom Model	0.7813	0.9939
Custom Model + *Dropout(0.25)*	0.8264	0.9945
Custom Model + *SpatialDropout(0.7)*	0.7617	0.9941
Custom Model + *DropBlock(bs = 3, kp = 0.9)*	0.8004	0.9931
Custom Model + *DropBlock(bs = 7, kp = 0.9)*	0.7956	0.9945

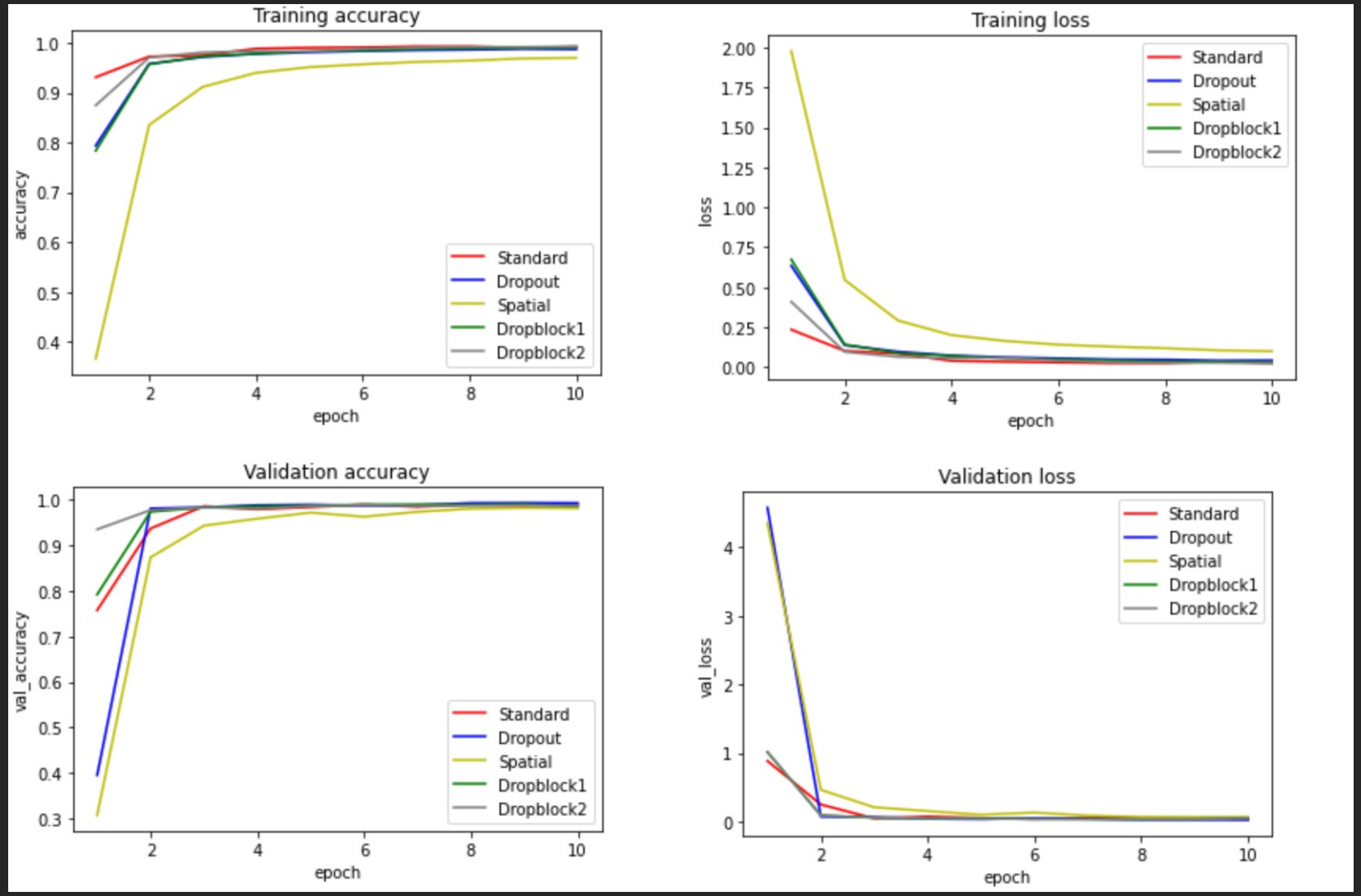
CustomNet accuracy



CustomNet performances

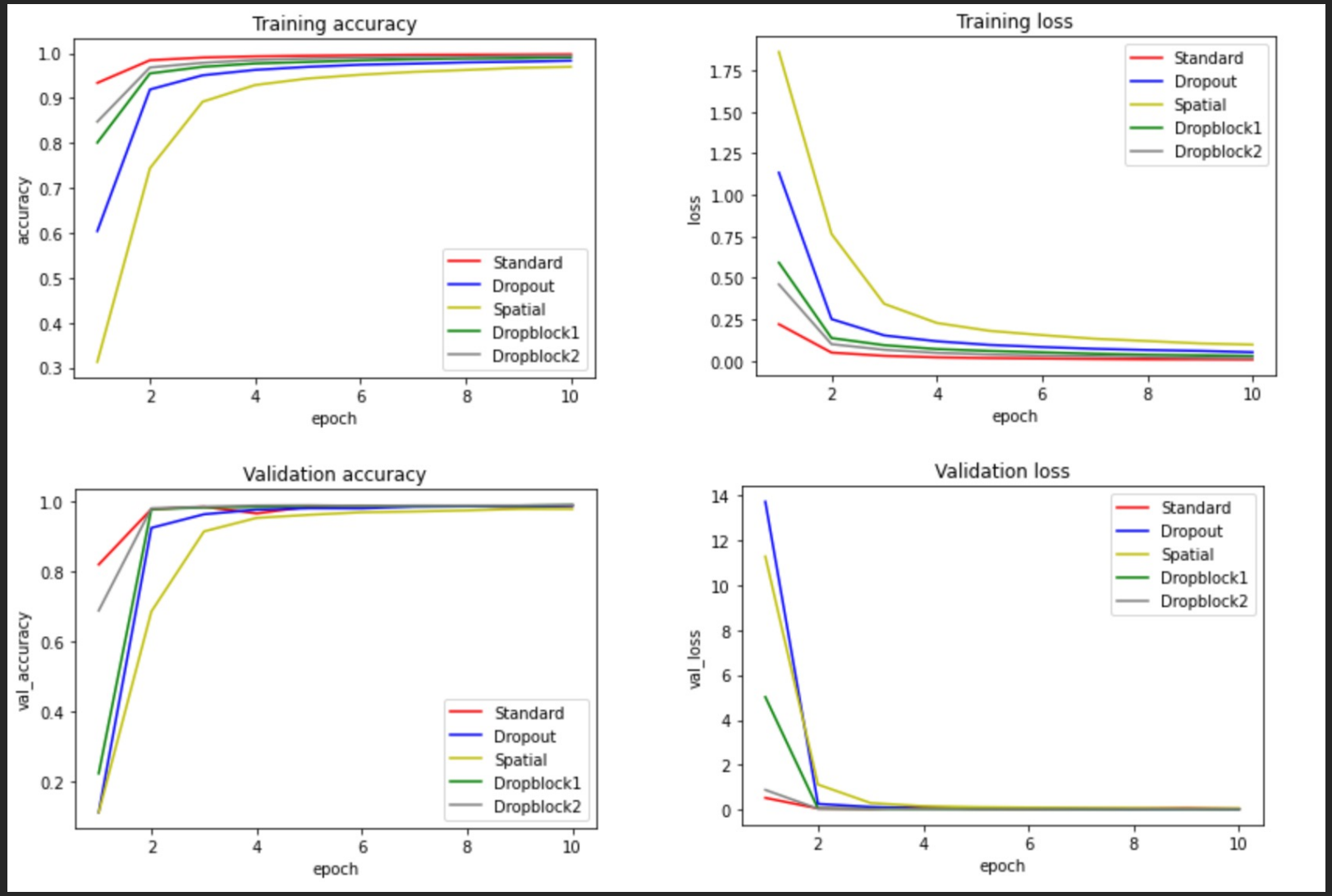
Ludovico Ottobre, Gianmarco Evangelista.

Results on MNIST



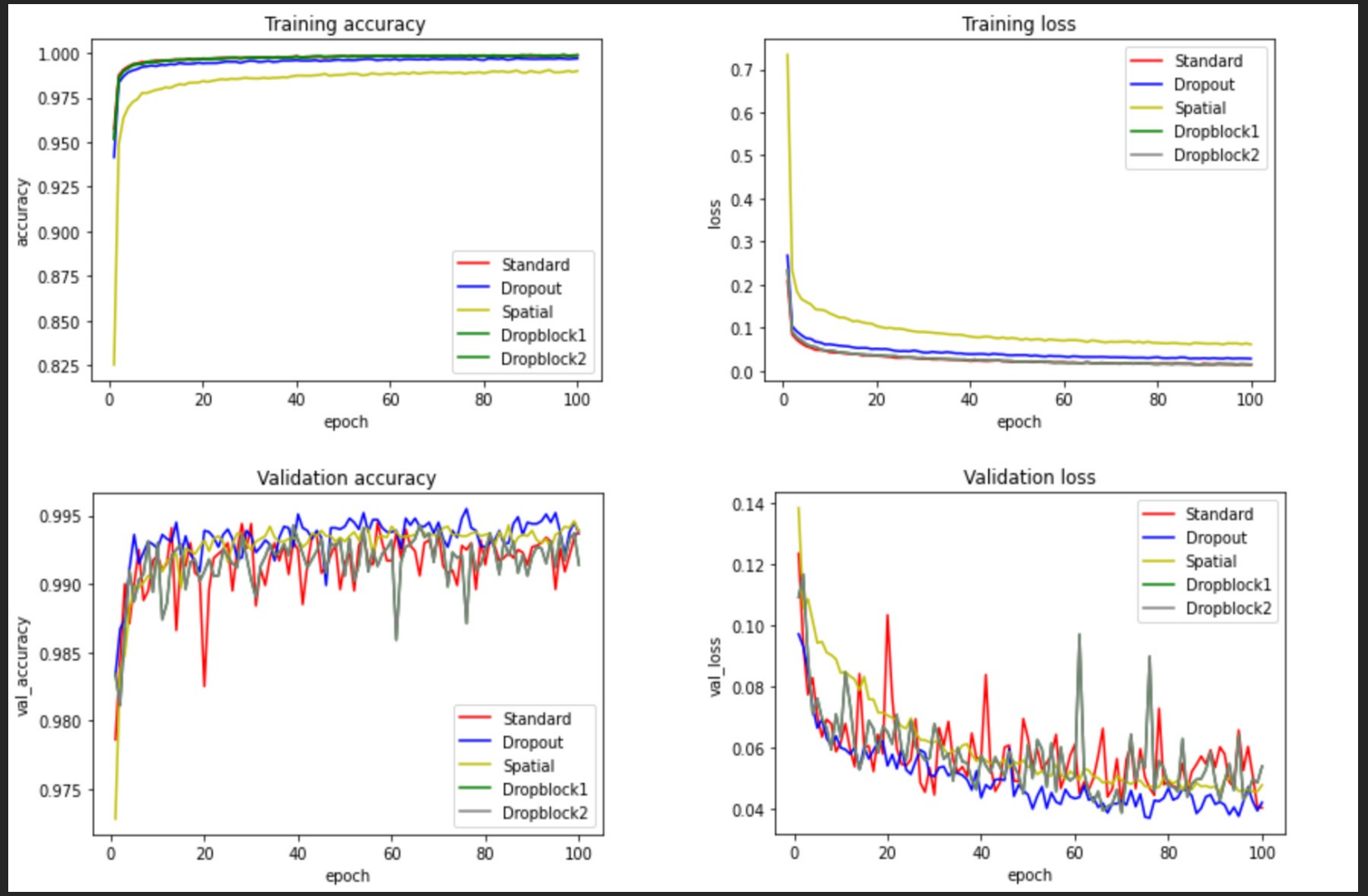
(a) DenseNet

Results on MNIST



(b) ResNet-50

Results on MNIST



(c) CustomNet

RetinaNet on ResNet-50 Experiment

Object detection is a computer vision technique that allows us to identify and locate objects in an image or video. With this kind of identification and localization, object detection can be used to count objects in a scene and determine and track their precise locations, all while accurately labelling them.

In order to implement and evaluate DropBlock performance also in this computer vision technique, we have used *RetinaNet* based on ResNet-50 and we have inserted inside ResNet-50 FPN the DropBlock method.

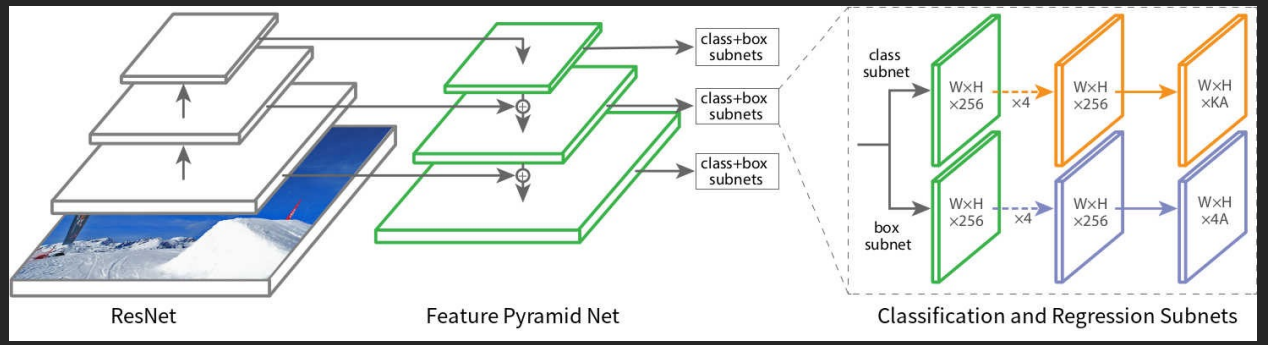
Ludovico Ottobre, Gianmarco Evangelista.

RetinaNet details

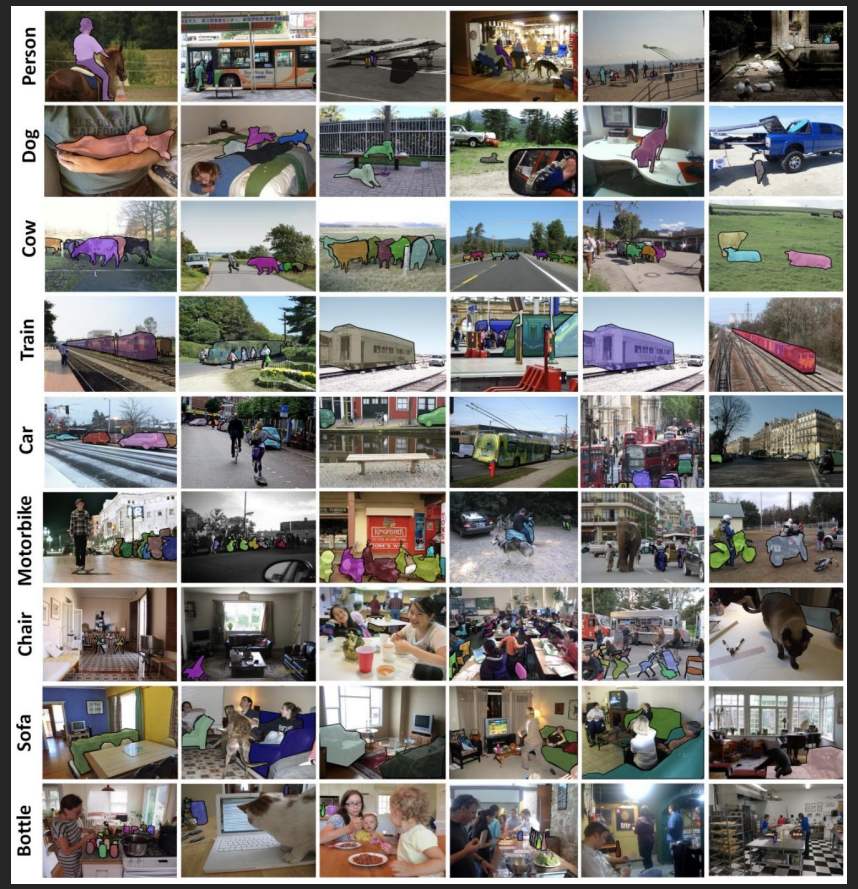
In this model, our goal is to localize and classify in the correct way all data into different categories. Object detection models can be broadly classified into "single-stage" and "two-stage" detectors. Two-stage detectors are often more accurate but at the cost of being slower. Here in this case, we have used *RetinaNet*, a single-stage detector, which is accurate and runs fast.

RetinaNet uses a *feature pyramid network* to efficiently detect objects at multiple scales and introduces a new loss, the *Focal loss function*, to alleviate the problem of the extreme foreground-background class imbalance.

Focal Loss gives to the model a bit more freedom to take some risk when making predictions. This is particularly important when dealing with highly imbalanced datasets beacuse in some cases, we really need to model to take a risk and predict something.



Object Detection Dataset



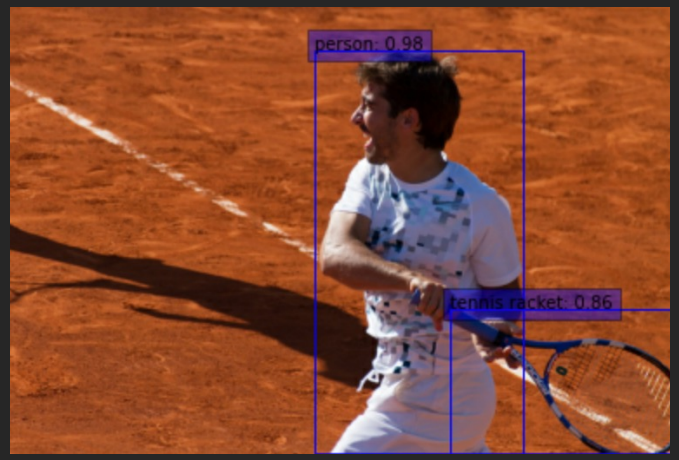
COCO-2017

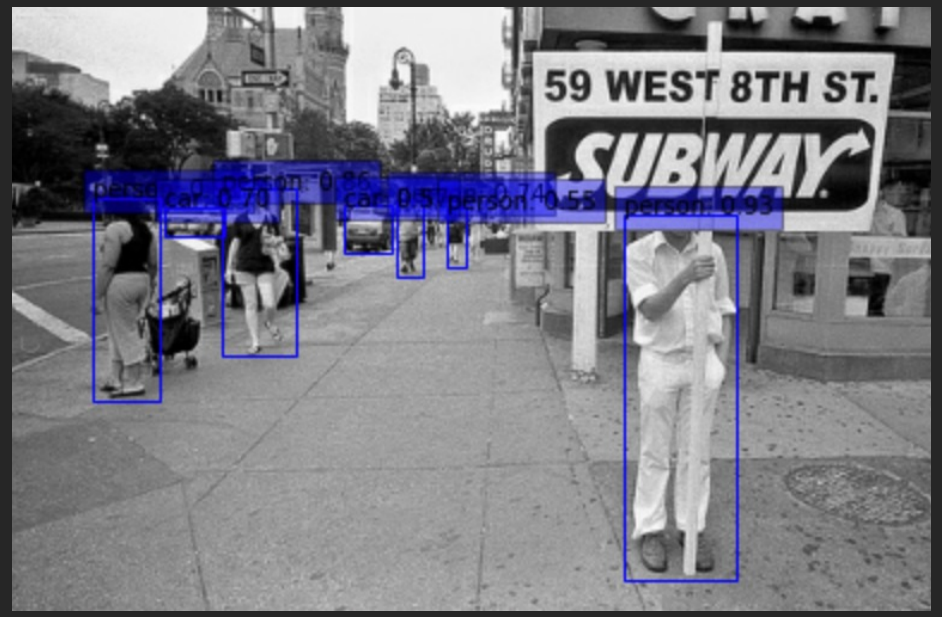
COCO is a large-scale object detection, segmentation, and captioning dataset. COCO has several features:

- Object segmentation
- Recognition in context
- Superpixel stuff segmentation
- 330K images (>200K labeled)
- 1.5 million object instances
- 80 object categories
- 91 stuff categories
- 5 captions per image

We have used a smaller subset of ~500 images for training in this case.

Object Detection with RetinaNet





RetinaNet on ResNet-50 FPN Experiment

MODEL	loss	val_loss
RetinaNet	1.6236	2.6221
RetinaNet + *Dropout(0.25)*	1.6299	2.5462
RetinaNet + *SpatialDropout(0.7)*	1.8380	2.6808
RetinaNet + *DropBlock(bs = 7, kp = 0.7)*	1.7043	2.5539
RetinaNet + *DropBlock(bs = 7, kp = 0.9)*	1.6841	2.5540
RetinaNet + *DropBlock(bs = 3, kp = 0.7)*	1.6414	2.6322
RetinaNet + *DropBlock(bs = 3, kp = 0.9)*	1.6985	2.6010

Analyzing the results based on *loss* and *val_loss*, we can say that lower is the loss, better is the model. The loss is calculated on training and validation and its interpretation is how well the model is doing for these two sets. Unlike accuracy, loss is not a percentage. It is a summation of the errors made for each example in training or validation sets. Loss value implies how well or poorly a certain model behaves after each iteration of optimization.
Ideally, one would expect the reduction of loss after each, or several, iteration(s). We considered the lowest loss value for each model and, making a theoretical evaluation of the results, we can see how DropBlock turns out to be a good alternative to other regularization methods such as Spatial and Dropout.

Ludovico Ottobre, Gianmarco Evangelista.

Conclusions

The DropBlock technique been evidenced to effectively beat some of the best performance results obtained by traditional Dropout and Spatial Dropout as well as shows better performance corresponding to strong data augmentation techniques. DropBlock has proved to be effective regularization approach for object detection. Our experiments show that applying DropBlock in skip con- nections in addition to the convolution. It is a form of structured dropout that drops spatially correlated information.

We demonstrate DropBlock is a more effective regularizer compared to Dropout in CIFAR-10 and MNIST classification, in the same way results to be a good alternative for object detection in COCO dataset.