Foundations of Deep Learning – Images classification on CIFAR10

PROJECT MADE BY DEIANA (844668), BIDONE (892054), RABIE (898348)

CODE:

https://colab.research.google.com/drive/1h9TfJHFaWj_2ZAuKLsl_0eG4NcgV6blZ?usp=sharing

MODEL:

https://drive.google.com/file/d/1nxj4nyTGF9csUZEl2-7exPYvB12QhdoB/view?usp=drive_link

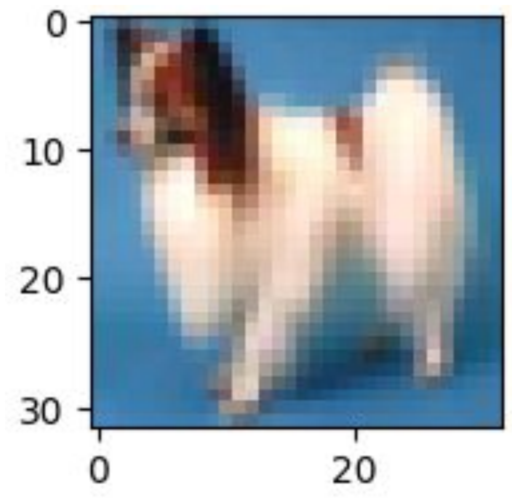
CIFAR-10 DATASET

It contains 60,000 color images of size 32-by-32 pixels

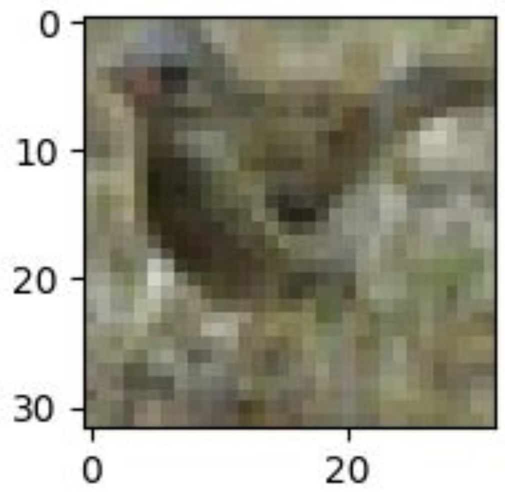
10 classes:
airplane, *automobile*, *bird*, *cat*, *deer*, *dog*, *frog*, *horse*, *ship*, and *truck*

Source:
https://www.mathworks.com/help/deeplearning/ug/data-sets-for-deep-learning.html

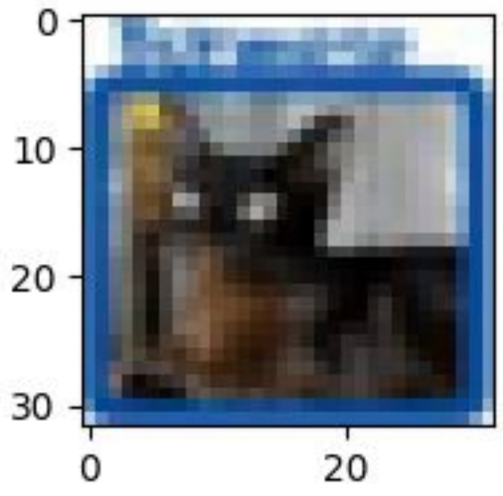
SOME IMAGES FROM THE DATASET
low resolution



Dog



Bird



Cat

IMPORTS AND INITIAL CONFIGURATIONS

01

Imported **libraries**:
- TensorFlow e Keras
- Numpy
- Random
- Os

02

Fix the **seed** for different operations:
Seed = 42

03

A function to detect available **hardware** and configure TensorFlow to use it

Loading the CIFAR-10 **dataset**

Conversion of integer labels into **binary vectors**(one-hot encoding):
0 □ *wrong class*
1 □ *correct class*

Image normalization:
Pixel values are converted from 0-255 to 0-1
Images are converted to float32 values
Subtracting the mean and dividing by the standard deviation

PREPROCESSING

DATA AUGMENTATION
for the training dataset

Data augmentation is a technique for improving model generalization by generating random variants of training images.

Random_flip_left_right □
flips images horizontally

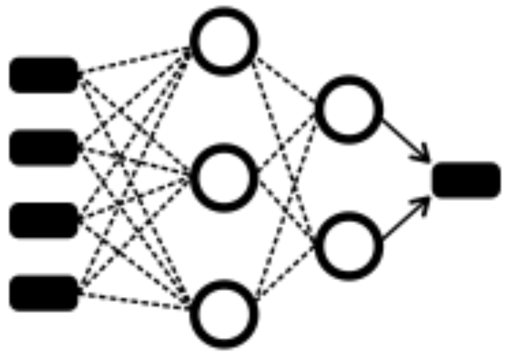
Reflected padding □
Adds a 2 pixel border around the image using reflected padding

Random crop □
Crop the image to a size of 32x32 pixels at a random location.

PERSONALIZED INITIALIZATION FOR WEIGHTS

A custom initializer for the convolution weights based on a **whitening method**:

- The training images and **patch size** are selected.
- Small squares are extracted from each image, as crops (patches).
- We calculate the **covariance matrix** to see how similar the patches are to each other.
- We used **eigenvalues** and **eigenvectors** of the matrix. Which are normalized and sorted to put the most important directions of change first and reduce variance.
- We use this information to create initial weights that are well balanced and represent the structure of the data well.



PERSONALIZED FUNCTIONS

Loss function (Categorical Crossentropy) with Label Smoothing:
label Smoothing helps the model generalize more, be less confident in its predictions, and improve overfitting. Instead of assigning 0 or 1, apply a small portion of the probability to all classes.

Activation function APTx:
APTx, proposed by Ravin Kumar (2022), is an activation function designed to improve deep learning efficiency. It requires fewer mathematical operations than MISH, speeding up training and reducing hardware needs. APTX maintains high performance, making it ideal for resource-constrained environments.

MODEL CREATION

Seven Convolutional layers Conv2D:

- First layer: *24 2x2 size filters, 32x32 size images with 3 channels (RGB).*
- Second-third layer: *128 3x3 size filters.*
- Fourth and fifth layer*: 256 3x3 size filters.*
- Sixth and seventh*: 512 3x3 size filters.*

Activation layers □ APTx custom activation function

Four Pooling layers □ MaxPooling2D, Reduces the spatial dimension of the input. The first three have a spatial dimension of 2x2, the last 3x3

Batch normalization □ normalizes activations for each batch, with manual momentum.

Flatten layer □ Flattens the input to prepare it for the final dense layer

Fully connected layer □ Dense to do the final classification, with softmax activation function

MODEL TRAINING

Created and compiled □ **Adam optimizer** & custom **loss function**

Callback:

- **ModelCheckpoint** saves the best model based on validation performance
- **EarlyStopping** stops training if validation performance does not improve for a specified number of epochs.
- **ReduceLRonPlateau** reduces LR if validation performance stops improving.

- **LearningRateScheduler** applies a LR scheduler:
  Is a personalized callback that progressively reduces the learning rate.

  High initial learning rate > ***0.0015***
  Then reduction to stabilize learning > ***0.0012***
  Final phase with low learning rate > ***0.0005***
- **CustomProgbarLogger** uses custom logger to monitor the LR:
  Is a personalized callback to monitor and record the learning rate at the end of each epoch. it is important to understand how model training is evolving.

TRAINING RESULTS – 10 epochs only

EPOCH 1/10

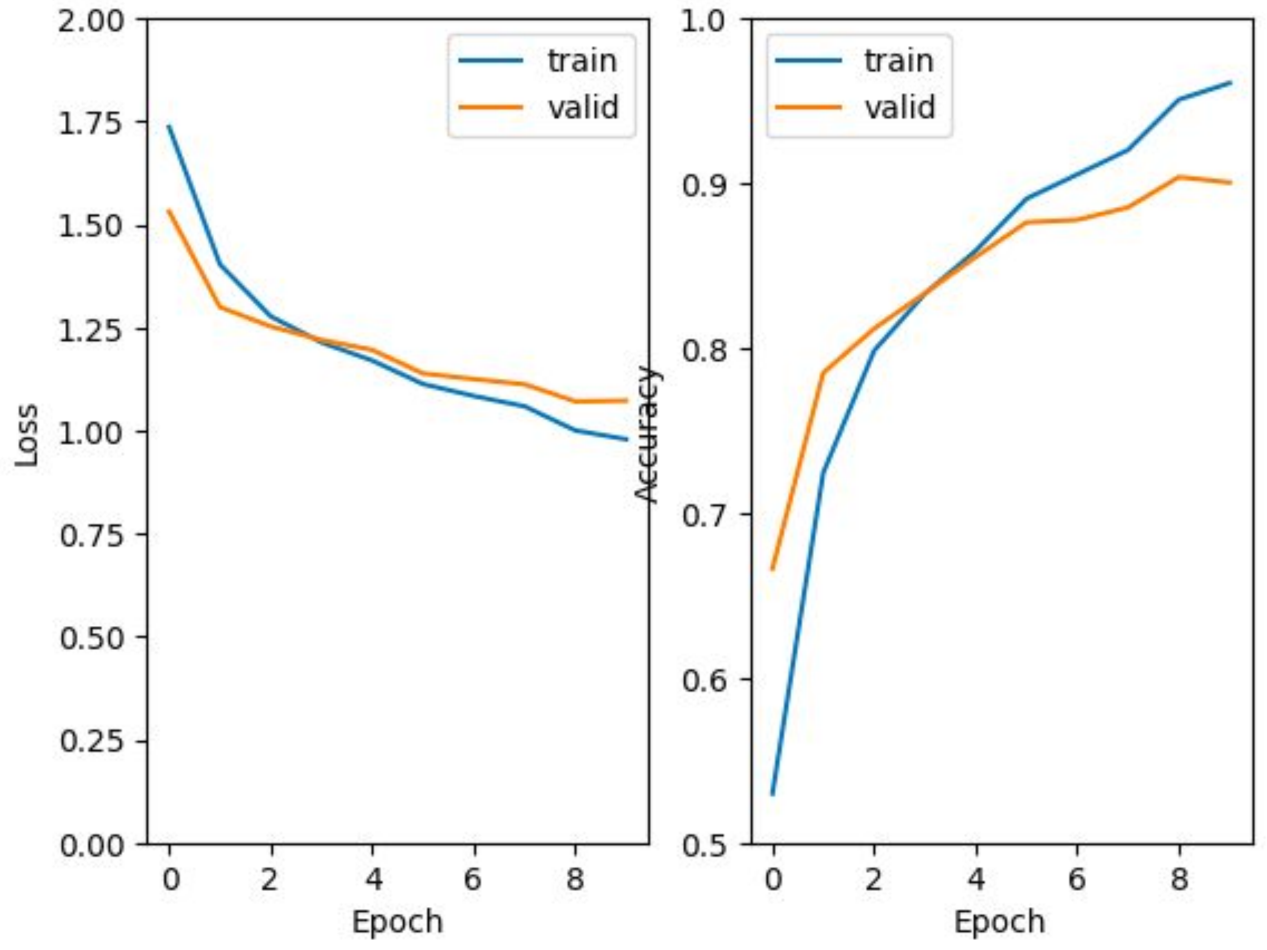
Training Loss: 1.7666
Training Accuracy: 51.85%

Validation Loss: 1.5789
Validation Accuracy: 64.96%

EPOCH 10/10

Training Loss: 0.9861
Training Accuracy: 95.80%

Validation Loss: 1.0761
Validation Accuracy: 90.14%



TEST SET ACCURACY

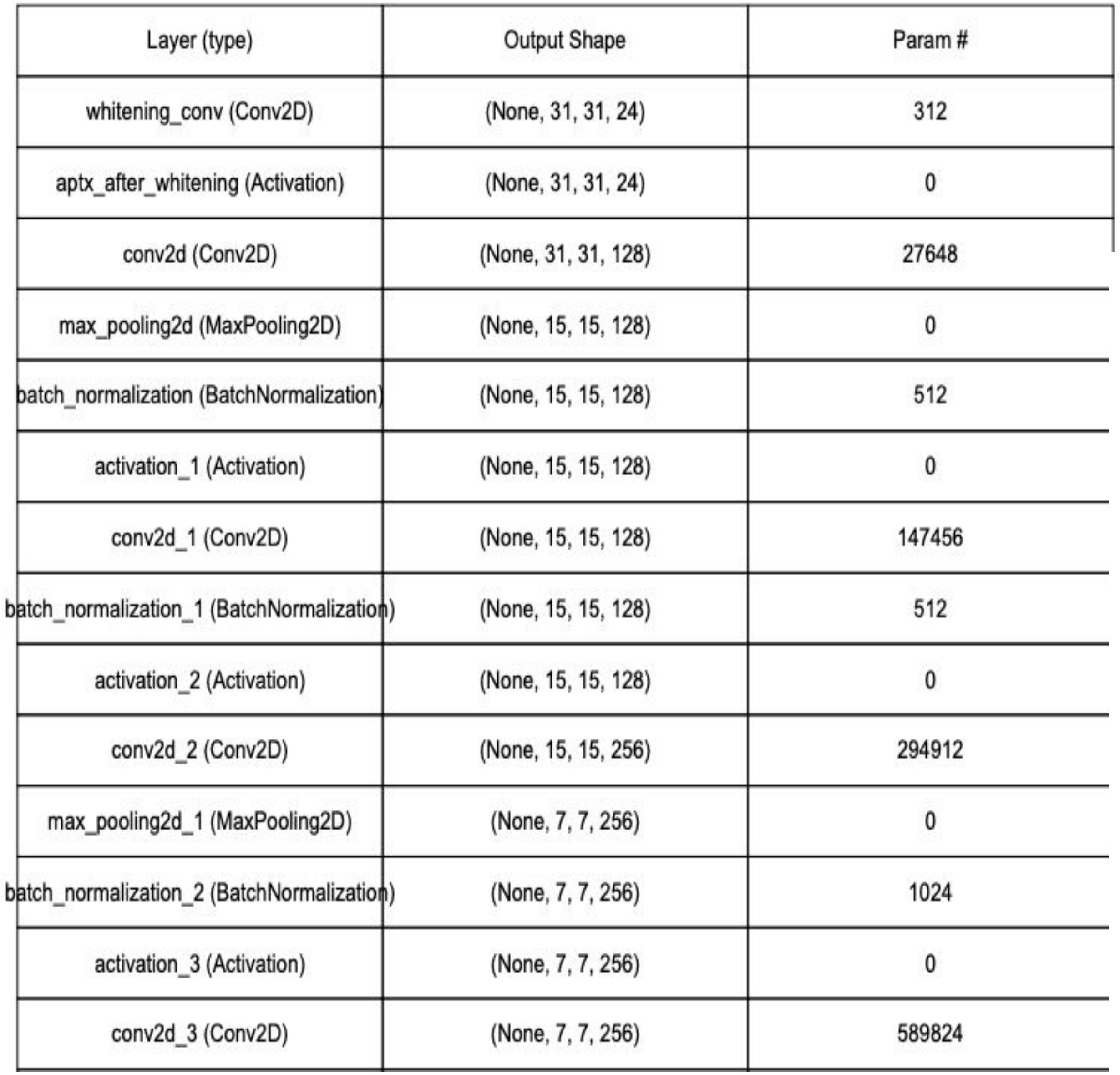
Final value of the accuracy: **91.13%** ✓

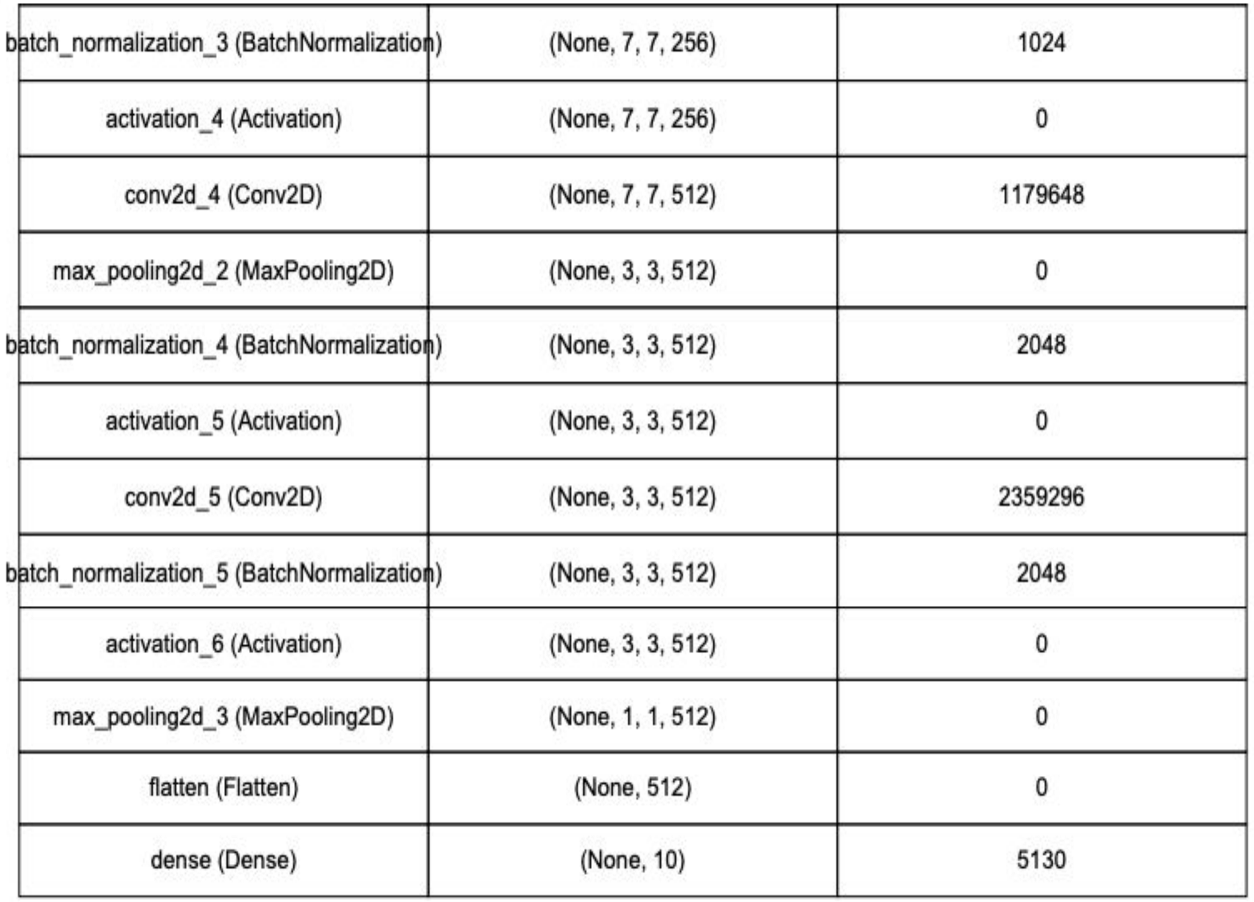
Batch Predictions Make predictions for each batch of test images, using the *augmentation function* during testing

Test_time_augmentation Augmentation function – It applies different transformations to test images (*flip, padding, cropping*) and combine the prediction results. This approach improves the robustness of the model by evaluating each image in different variations.

Test accuracy Calculate the average accuracy of the model across all test batches.

SUMMARY





Layer (type)	Output Shape	Param #
whitening_conv (Conv2D)	(None, 31, 31, 24)	312
aptx_after_whitening (Activation)	(None, 31, 31, 24)	0
conv2d (Conv2D)	(None, 31, 31, 128)	27648
max_pooling2d (MaxPooling2D)	(None, 15, 15, 128)	0
batch_normalization (BatchNormalization)	(None, 15, 15, 128)	512
activation_1 (Activation)	(None, 15, 15, 128)	0
conv2d_1 (Conv2D)	(None, 15, 15, 128)	147456
batch_normalization_1 (BatchNormalization)	(None, 15, 15, 128)	512
activation_2 (Activation)	(None, 15, 15, 128)	0
conv2d_2 (Conv2D)	(None, 15, 15, 256)	294912
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 256)	0
batch_normalization_2 (BatchNormalization)	(None, 7, 7, 256)	1024
activation_3 (Activation)	(None, 7, 7, 256)	0
conv2d_3 (Conv2D)	(None, 7, 7, 256)	589824
batch_normalization_3 (BatchNormalization)	(None, 7, 7, 256)	1024
activation_4 (Activation)	(None, 7, 7, 256)	0
conv2d_4 (Conv2D)	(None, 7, 7, 512)	1179648
max_pooling2d_2 (MaxPooling2D)	(None, 3, 3, 512)	0
batch_normalization_4 (BatchNormalization)	(None, 3, 3, 512)	2048
activation_5 (Activation)	(None, 3, 3, 512)	0
conv2d_5 (Conv2D)	(None, 3, 3, 512)	2359296
batch_normalization_5 (BatchNormalization)	(None, 3, 3, 512)	2048
activation_6 (Activation)	(None, 3, 3, 512)	0
max_pooling2d_3 (MaxPooling2D)	(None, 1, 1, 512)	0
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 10)	5130

LIMITATIONS OF THE MODEL AND ALTERNATIVE APPROACHES

During the construction of the model, various techniques and combinations of hyperparameters were tested which proved to be suboptimal.

> **Dropout**: randomly selected neurons are ignored during training. This technique hindered learning even with low values, and did not particularly favor generalization, so it was eliminated.

> **More convolutional layers and more filters**: in this case we were able to obtain better levels of accuracy, but the execution time increased dramatically, and we considered it not worth it.

> **More epochs**: we noticed a plateau in the accuracy on the validation set around 90/91% with a high number of epochs, this indicates that there is no point in continuing the training and that we would need to modify the model if we wanted to achieve a higher accuracy.

FINAL CONCLUSIONS

> High efficency of the model with only **10 Epochs**
> 72s of training time on single GPU

> Traning accuracy **95.80%**
> Validation accuracy **90.14%**
> Test accuracy **91.13%**

> Avoid overfitting and underfitting thanks to:

- *Accurate data normalization*
- *Well-designed model architecture with convolution, pooling and batch normalization layers.*
- *Using nonlinear activation functions.*
- *Accurate training setup and monitoring.*

FUTURE IMPROVEMENTS

> **Triangular program** for learning rate

> **Hyperparameter search**: Use techniques such as random or Bayesian search to find the best values for hyperparameters.

> **More advanced optimizer**: You can use a more advanced optimizer to achieve faster convergence, such as AdamW, an extension of Adam that includes weight decay regularization to improve generalization. Additionally, you can add Lookahead, a wrapper that uses two optimizers, periodically updating the weights with the average of the 'explorers' updates to stabilize convergence and accelerate optimization.