IMAGE CLASSIFICATION

USING CIFAR-10 AND CIFAR-100 DATASETS

Elisa Du
Smart Technologies
December 17, 2023

Table of Contents

1. Introduction

The primary objective of the project is to build a convolutional neural network model that can classify between twenty-four different classes of the CIFAR-10 and CIFAR-100 datasets. The classes are car, bird, cat, deer, dog, horse, truck, cattle, fox, baby, boy, girl, man, woman, rabbit, squirrel, tree, bicycle, bus, motorcycle, pickup truck, train, lawn mower and tractor. This involves obtaining, preparing and exploring the data sets.

Several CNN models are then constructed and fine-tuned to achieve optimal classification performance. Hyperparameter tuning is guided by continuous evaluation of model accuracy. Our focus is not only on achieving high accuracy, but also on improving the fairness of model predictions, especially across classes with unbalanced representation.

The aim of this project is to present the CNN model that achieves the highest test accuracy, demonstrating its ability to accurately classify images across the diverse set of twenty-four classes.

2. Dataset

The underlying datasets can be downloaded from 'https://cs.toronto.edu/~kriz/cifar.html'. They consist of different classes, separated into training and test images with corresponding labels. Each training and test image is also divided into several batches.

2.1 CIFAR-10

The CIFAR-10 dataset consists of 50000 training and 10000 test 32x32 colour images, 10 classes with 5 training and 1 test batch. Each class contains 6000 images. Each batch contains randomly selected images from each class, while the training batches contain 10000 images each and the test batches contain 1000 images each. When the training batches are added together, there are exactly 5000 images from each class.

The following 10 classes are in the dataset:

- Airplane
- Automobile
- Bird
- Cat
- Deer
- Dog
- Frog
- Horse
- Ship
- Truck

2.2 CIFAR-100

Similar to the CIFAR-10 dataset, this dataset also contains classes of images. There are 100 classes of 600 images each, divided into a five-to-one split, with 500 training images and 100 test images per class. Each class is grouped into a superclass. The data set contains 20 superclasses. The images are labelled with a 'fine' label for the exact class and a 'coarse' label for the superclass.

The classes contained are shown below:

Superclass	Classes
aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
food containers	bottles, bowls, cans, cups, plates
fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
household electrical devices	clock, computer keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

Table 1: Overview of CIFAR-100 classes

2.3. Combining CIFAR-10 with CIFAR-100

To extract the necessary classes for the classification problem, we first need to find the labels associated with the searched label names. For CIFAR-10, this information is stored in a separate file 'batches.meta' as a dictionary. For CIFAR-100, the meta file contains only the label names stored as lists, separated between the fine labels and the coarse label names as a dictionary. So we define number labels by the index of the names.

We concentrate first on getting all the images with the associated labels stored in separate lists. This is achieved by iterating through the batches and extracting the necessary information. For CIFAR-100, the fine and coarse labels are also included. The required classes also include a superclass ‘tree’, so we replace all fine labels that have the corresponding class with a new identical label.

We then filter the lists by the required labels and combine the two datasets by first remapping the labels and appending one to the end of the other dataset. The remapping is necessary to avoid duplication of labels when combining the datasets.

The resulting dataset consists of 45,500 images. 7 classes (from CIFAR-10) contain 5000 images each, the required superclass from CIFAR-100 has 2500 images and all other classes have 500 images each. All images are implemented as 3072-pixel bytes.

3. Data Preprocessing

Looking at the distribution of the underlying dataset, the classes are highly unbalanced. Therefore, we downsample the classes with highly significant images. The limit is set at 3000 images per class. In addition, the images are transformed into 32x32 RGB images. These images are displayed at a 90-degree angle. This introduces a compensating rotation to rotate them back.

Smaller classes are handled by data augmentation, which is applied by the batch generator in training. In addition, we implement an oversampling function that applies image augmentation to smaller classes in the training dataset to increase the number of such images. This augmentation consists of random flipping, cropping, brightening and contrast (Code 3). For downsampling, we will reduce each class in such a way that not 50% of the data per class is lost. In oversampling, we will only double the data, ensuring that too many new synthetic images are not generated, and real data continues to hold value.

Images are converted to greyscale. This allows the shape of the images to be highlighted. As with the model tested later, adding contrast does not improve accuracy. A saturation of 1.5 shows a slight increase of 0.06% in test accuracy. As it doesn't have a negative impact, we will keep this saturation in the data preprocessing.

4. Training

The following section describes how the model architecture is selected and what measurements are taken to optimize the best model and minimize overfitting. The model and batch generator are first selected without oversampling for the minor classes. The fairness of the model is then evaluated and measurements such as oversampling are applied.

4.1 Model Selection

For the model, we recreated different model architectures that showed good results on the CIFAR-10 dataset and adapted them to the existing code. Thus, each of the models uses the Adam optimizer with a learning rate of 0.001 and cross entropy cost as loss function. It is applied to grayscale images with an input shape of 32x32x1. All models run with an epoch number of 40 and a batch size of 64. The size of the batch is chosen based on the best accuracy after testing on the LeNet model.

The best model is selected by the accuracy of the validation and test data. Table 1 shows the results.

Model	Validation Accuracy	Test Accuracy
LeNet model (Loane, 2023)	56.22%	60.41%
Modified LeNet model (Loane, 2023)	48.78%	52.64%
4-layer CNN (Płotka, 2018)	60,78%	64,20%
6-layer CNN (Płotka, 2018)	62.67%	65.63%
8-layer CNN	59.25%	62.91%
6-layer CNN with Batch Normalization (Brownlee, 2020)	65.75%	68.91%
4-layer CNN with Batch Normalization (Chansung, 2018)	62.24%	65.98%

Table 2: Validation and test accuracy of base models

The 6-layer CNN with batch normalization shows the most promising results for our dataset based on these accuracy values. It is built on the architecture of the VGG models introduced by the Visual Geometry Group at the University of Oxford (Zisserman, 2015). It is designed for image classification tasks and consists of stacking multiple convolutional layers with small 3x3 filters followed by max-pooling layers. The model already uses dropout regularization and batch normalization to stabilize and speed up the learning process (Brownlee, 2020).

```
def model1():
  model = Sequential()
  model.add(Conv2D(32, (3, 3), activation='relu', padding='same',
input_shape=(32, 32, 3)))
  model.add(BatchNormalization())
  model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
  model.add(BatchNormalization())
  model.add(MaxPooling2D((2, 2)))
  model.add(Dropout(0.5))
  model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
  model.add(BatchNormalization())
  model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
  model.add(BatchNormalization())
  model.add(MaxPooling2D((2, 2)))
  model.add(Dropout(0.5))
  model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
  model.add(BatchNormalization())
  model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
  model.add(BatchNormalization())
  model.add(MaxPooling2D((2, 2)))
  model.add(Dropout(0.5))
  model.add(Flatten())
  model.add(Dense(128, activation='relu'))
  model.add(BatchNormalization())
  model.add(Dropout(0.5))
  model.add(Dense(num_classes, activation='softmax'))
  model.compile(Adam(learning_rate=0.0015), loss='categorical_crossentropy',
metrics=['accuracy'])
  return model
```

Code 1: 6-layer CNN Base Model

4.2 Batch Generator

To handle the unbalanced data set, we use a batch generator with data augmentation. Two different methods are tested. The first, which is also used in the model selection, is the ImageDataGenerator provided by TensorFlow (Tensorflow, 2023). It randomly adds horizontal and vertical shifts of up to 10% of the total width and height of the images for each batch. The images are also randomly zoomed by up to 20%, sheared and rotated by up to 10 degrees.

```
datagen = ImageDataGenerator(width_shift_range=0.1, height_shift_range = 0.1,
zoom_range = 0.2, shear_range = 0.1, rotation_range=10)
```

Code 2: Implementation of a Image Data Generator in TensorFlow

The second generator is a self-defined batch generator. It iterates through the training dataset up to the batch size and randomly applies a data augmentation to the images. With a probability of 50%, the image is randomly cropped after being resized to 38x38. The image is then randomly flipped vertically, brightened and contrast adjusted.

```
def random_augment(image):
  if np.random.rand() < 0.5:
    image = tf.image.resize(image, [38, 38],
method=tf.image.ResizeMethod.NEAREST_NEIGHBOR)
    image = tf.image.random_crop(image, size=(32, 32, 3))
  image = tf.image.random_flip_left_right(image)
  image = tf.image.random_brightness(image, 0.1, 111)
  image = tf.image.random_contrast(image, 1, 1.5,111)
  return image
```

Code 3: Implementation of a random image augmentation in TensorFlow

In the test of both generators, the first has slightly better results with a difference of 0.82%. Furthermore, the epoch times are six times better and the training is more stabilized than with the other generator. For further optimization of the hyperparameters, we therefore use the ImageDataGenerator provided.

4.3 Color Spaces

In our previous experiments, we observed that using RGB images instead of grayscale images resulted in an improvement in accuracy, demonstrating an increase of almost 2%. The test accuracy reached 70.88%. This finding aligns with the insights shared in the paper "ColorNet: Investigating the Importance of Color Spaces for Image Classification" by Shreyank N Gowda in 2019. According to the paper, the use of various color spaces in the CIFAR-10 dataset with a simple CNN did not yield significant differentiation in performance (Shreyank N Gowda, 2019). Consequently, we have decided to continue using the RGB color scheme for our images, considering its demonstrated enhancement in accuracy compared to grayscale images.

4.4 Learning Rate

The learning rate was initially set to 0.001. This is compared to setting the learning rate to 0.002 and 0.0015. With the initial learning rate, the model seems to converge faster. However, the test accuracy improved to 71.99% with learning rate=0.0015, whereas the accuracy was almost the same with learning rate=0.001 and learning rate=0.002. For further testing the learning rate of 0.0015 was therefore chosen.

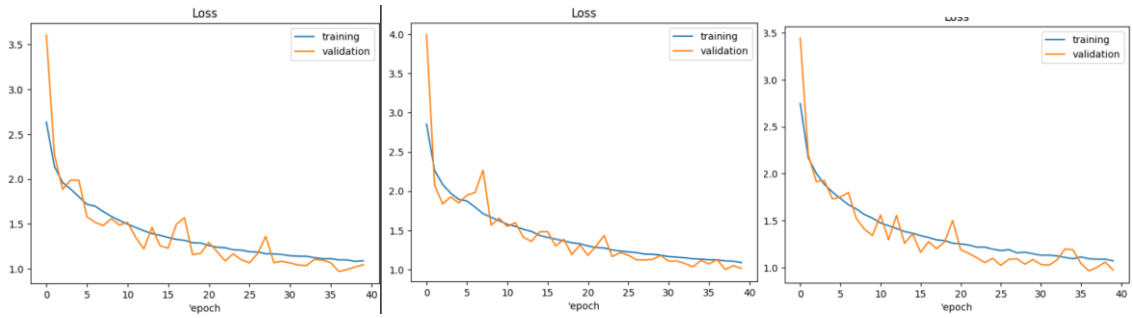

Figure 1: Training and validation loss with learning rate=0.002 (left), learning rate=0.001 (middle), learning rate=0.0015 (right)

4.5 Model Fairness

To address the issue of class imbalance in our dataset, we implemented an oversampling strategy for the minority classes during the training process, in addition to using data augmentation techniques with the ImageDataGenerator. Initially, our dataset presented a significant imbalance, with around 2000 images per class for the first eight classes and less than 500 images per class for the remaining classes.

After introducing the oversampling function, the overall accuracy of our model dropped to 54.82% after 40 epochs. When we relied solely on image augmentation to oversample the minor classes (without the ImageDataGenerator), we observed a significant increase in accuracy of 7%.

To further improve the performance of the model and mitigate the loss of accuracy, we have expanded the architecture of our convolutional network. The model now consists of three sets of 32-filter convolutional layers, four sets of 64-filter convolutional layers, and four sets of 128-filter convolutional layers. Each set was followed by max-pooling (2x2), dropouts and batch normalization.

Despite this architectural improvement, the overall accuracy of the model dropped from 71.99% to 68.88%. However, there was a positive result in the form of improved recall scores for the minority classes. This indicates that while the overall accuracy may have slightly decreased, the model's ability to correctly identify instances from the underrepresented classes has significantly improved, reflecting a more balanced and equitable performance across all classes.

Class	Accuracy of old model without oversampling	Accuracy of new model with oversampling
Automobile [0]	88.70%	80.50%
Bird [1]	73.60%	72.30%
Cat [2]	55.90%	54.50%
Deer [3]	84.50%	81.40%
Dog [4]	72.60%	56.10%
Horse [5]	84.50%	72.80%
Truck [6]	84.50%	75.00%
Trees [7]	96.60%	92.60%
Baby [8]	40.00%	54.00%
Bicycle [9]	86.00%	83.00%
Boy [10]	33.00%	24.00%
Bus [11]	25.00%	20.00%
Cattle [12]	08.00%	18.00%
Fox [13]	29.00%	34.00%
Girl [14]	25.00%	21.00%
Lawn Mower [15]	72.00%	80.00%
Man [16]	26.00%	26.00%
Motorcycle [17]	96.00%	85.00%
Pickup Truck [18]	21.00%	42.00%
Rabbit [19]	09.00%	20.00%
Squirrel [20]	17.00%	19.00%
Tractor [21]	64.00%	65.00%
Train [22]	67.00%	66.00%
Woman [23]	23.00%	37.00%

Table 3: Comparison of accuracy for each class between the non-oversampled and oversampled models

4.6 Batch Size

Looking at the expanded architecture, the batch size and learning rate remain the same as other alternatives lead to over- or underfitting.

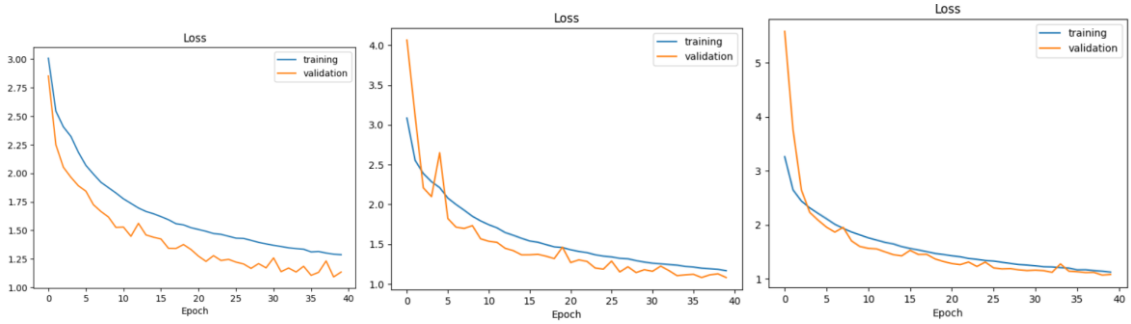

Figure 2: Training and Validation loss with batch size=16 (left), batch size=32 (middle) and batch size=64 (right)

5. Model evaluation

The final model in our experiments is an 11 convolutional layer architecture. This model includes three sets of 32-filter convolutional layers, four sets of 64-filter convolutional layers, and another four sets of 128-filter convolutional layers. Each convolutional layer has a kernel size of 3x3, and batch normalization is applied after each layer. A 2x2 max-pooling layer follows each change in filter size. To mitigate overfitting, we've included dropout layers with a dropout rate of 0.5 after every batch normalization.

Following the convolutional layers, a flatten layer is introduced to transform the multi-dimensional output into a one-dimensional array. This is followed by two dense layers of 128 and 24 neurons respectively. The model has a total of 934,616 parameters, of which 932,632 are trainable. Table 3 and Figure 3 show the complete architecture and implementation.

The Adam optimizer is used with a learning rate of 0.0015 and the cross-entropy loss function is used for training. The batch size is set to 64, so that the training process consists of 469 steps per epoch. Before each epoch, the training data is shuffled to improve the generalization ability of the model.

We performed downsampling on each class, limiting the number of images to a maximum of 2500 per class. Following the training and validation split, which was set at 60% for training and 30% for validation, we then applied oversampling to all classes, ensuring a minimum of 1000 images per class. This aims to address class imbalance in the dataset.

The test accuracy of the model is 69.05% after 100 epochs. It shows difficulties in predicting person types (gender, adult/child) and in animals. High accuracy reached the model especially in the classes, which are already major represented in the dataset. Detailed metrics are provided in Table 5.

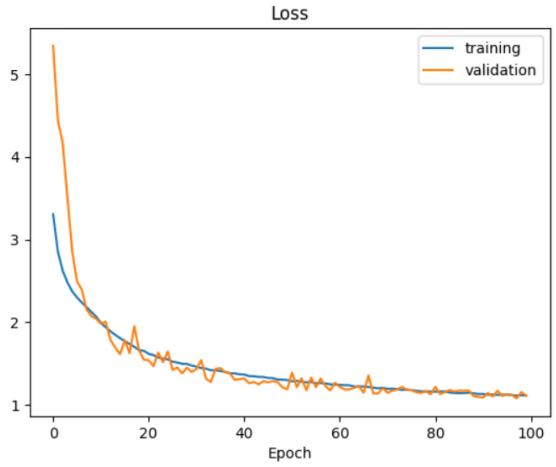

Figure 3: Loss function of the final model after 100 epochs

Layer (type)	Output shape	Param #
Conv2D	(None, 32, 32, 32)	896
Batch Normalization	(None, 32, 32, 32)	128
Dropout	(None, 32, 32, 32)	0
Conv2D	(None, 32, 32, 32)	9249
Batch Normalization	(None, 32, 32, 32)	128
Dropout	(None, 32, 32, 32)	0
Conv2D	(None, 32, 32, 32)	9249
Batch Normalization	(None, 32, 32, 32)	128
Max Pooling2D	(None, 16, 16, 32)	0
Dropout	(None, 16, 16, 32)	0
Conv2D	(None, 16, 16, 64)	18496
Batch Normalization	(None, 16, 16, 64)	256
Dropout	(None, 16, 16, 64)	0
Conv2D	(None, 16, 16, 64)	36928
Batch Normalization	(None, 16, 16, 64)	256
Dropout	(None, 16, 16, 64)	0
Conv2D	(None, 16, 16, 64)	36928
Batch Normalization	(None, 16, 16, 64)	256
Dropout	(None, 16, 16, 64)	0
Conv2D	(None, 16, 16, 64)	36928
Batch Normalization	(None, 16, 16, 64)	256
Max Pooling2D	(None, 8, 8, 64)	0
Dropout	(None, 8, 8, 64)	0
Conv2D	(None, 8, 8, 128)	73856
Batch Normalization	(None, 8, 8, 128)	512
Dropout	(None, 8, 8, 128)	0
Conv2D	(None, 8, 8, 128)	147584
Batch Normalization	(None, 8, 8, 128)	512
Dropout	(None, 8, 8, 128)	0
Conv2D	(None, 8, 8, 128)	147584
Batch Normalization	(None, 8, 8, 128)	512
Dropout	(None, 8, 8, 128)	0
Conv2D	(None, 8, 8, 128)	147584
Batch Normalization	(None, 8, 8, 128)	512
Max Pooling	(None, 4, 4, 128)	0
Dropout	(None, 4, 4, 128)	0
Flatten	(None, 2048)	0
Dense	(None, 128)	262272
Batch Normalization	(None, 128)	512
Dropout	(None, 128)	0
Dense	(None, 24)	3096

Table 4: Model architecture

```
def model1():
  model = Sequential()
  model.add(Conv2D(32, (3, 3), activation='relu', padding='same',
input_shape=(32, 32, 3)))
  model.add(BatchNormalization())
  model.add(Dropout(0.5))
  model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
  model.add(BatchNormalization())
  model.add(Dropout(0.5))
  model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
  model.add(BatchNormalization())
  model.add(MaxPooling2D((2, 2)))
  model.add(Dropout(0.5))
  model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
  model.add(BatchNormalization())
  model.add(Dropout(0.5))
  model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
  model.add(BatchNormalization())
  model.add(Dropout(0.5))
  model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
  model.add(BatchNormalization())
  model.add(Dropout(0.5))
  model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
  model.add(BatchNormalization())
  model.add(MaxPooling2D((2, 2)))
  model.add(Dropout(0.5))
  model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
  model.add(BatchNormalization())
  model.add(Dropout(0.5))
  model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
  model.add(BatchNormalization())
  model.add(Dropout(0.5))
  model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
  model.add(BatchNormalization())
  model.add(Dropout(0.5))
  model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
  model.add(BatchNormalization())
  model.add(MaxPooling2D((2, 2)))
  model.add(Dropout(0.5))
  model.add(Flatten())
  model.add(Dense(128, activation='relu'))
  model.add(BatchNormalization())
  model.add(Dropout(0.5))
  model.add(Dense(num_classes, activation='softmax'))
  model.compile(Adam(learning_rate=0.0015), loss='categorical_crossentropy',
metrics=['accuracy'])
  return model
```

Code 4: Implementation of the final model in TensorFlow

Class	Precision	Recall	F1-Score	Support
Automobile [0]	0.86	0.91	0.88	1000
Bird [1]	0.64	0.75	0.69	1000
Cat [2]	0.54	0.60	0.57	1000
Deer [3]	0.70	0.75	0.72	1000
Dog [4]	0.64	0.65	0.64	1000
Horse [5]	0.84	0.74	0.79	1000
Truck [6]	0.85	0.79	0.82	1000
Trees [7]	0.65	0.98	0.78	500
Baby [8]	0.48	0.39	0.43	100
Bicycle [9]	0.83	0.65	0.73	100
Boy [10]	0.43	0.32	0.37	100
Bus [11]	0.44	0.15	0.22	100
Cattle [12]	0.47	0.20	0.28	100
Fox [13]	0.52	0.33	0.40	100
Girl [14]	0.35	0.31	0.33	100
Lawn Mower [15]	0.84	0.67	0.74	100
Man [16]	0.49	0.27	0.35	100
Motorcycle [17]	0.89	0.73	0.80	100
Pickup Truck [18]	0.58	0.33	0.42	100
Rabbit [19]	0.28	0.11	0.16	100
Squirrel [20]	0.44	0.15	0.22	100
Tractor [21]	0.55	0.54	0.54	100
Train [22]	0.53	0.60	0.56	100
Woman [23]	0.36	0.26	0.30	100

Table 5: Metrics of the final model including precision, recall, f1-score and number of test data per class

6. List of Figures

7. List of Tables

8. List of Codes

9. References

Brownlee, J. (2020, August 28). *Machine Learning Mastery*. Retrieved 12 16, 2023, from https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/

Chansung, P. (2018, April 17). *Towards Data Science*. Retrieved 15 12, 2023, from https://towardsdatascience.com/cifar-10-image-classification-in-tensorflow-5b501f7dc77c

Loane, J. (2023, November). *Github*. Retrieved 12 15, 2023, from https://github.com/johnloane/st_23_sd4a/blob/master/GTS_23_sd4a.ipynb

Płotka, S. (2018, August 27). *Ermlab*. Retrieved 12 15, 2023, from https://ermlab.com/en/blog/nlp/cifar-10-classification-using-keras-tutorial/

Tensorflow. (2023, December 16). Retrieved from https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator

Zisserman, K. S. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. *https://arxiv.org/pdf/1409.1556.pdf*, 14.