Raport of Image classification (team: BIG EGGS)

GOAL: Build a machine learning model that can accurately classify images into different categories, such as animals, plants, or everyday objects. Use a dataset such as ImageNet or CIFAR-10 to train and test your model.

Overview of data

Dataset was used CIFAR-10, which is a collection of 50,000 training images and 10,000 test images, where each image belongs to one of 10 classes('airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck'). Per class is 6000 images(5000 and 1000). Images shape is 32x32. Link: https://paperswithcode.com/dataset/cifar-10

Normalizing data:
Data was normalized with rescale images the pixel values of the images from the range of [0, 255] to [0, 1](CNN model can benefit from improved numerical stability, better gradient management, and compatibility with various network architectures and activation functions). Labels was converted from a two-dimensional shape (matrix with one column and multiple rows) to a one-dimensional shape (vector) with flatten(), for sparse_categorical_crossentropy

For better model training was used data augmentation with using ImageDataGenerator. For artificially increase the size of the training dataset by applying random transformations to the existing images. It helps in improving the model's ability to generalize and handle variations in the input data.
Different parameters of ImageDataGenerator(random rotations is 15, range of random zooming is 0.1, range of random horizontal and vertical shifts is 0.1)

Methods used

Creating Convolutional Neural Network (CNN) model designed for image classification:

- The architecture consists of several layers: Conv2D, BatchNormalization, MaxPooling2D, Dense, Dropout, Flatten
- Conv2D performs 2D convolution on the input data. Takes 32, 64 or 128 number of filters, filter/kernel size (3x3 in this case). First Conv2D have input_shape=(32, 32, 3) of the input images that will be fed into the network. In all Conv2D layers used padding same that ensures the spatial dimensions (height and width) of the input and output feature maps remain the same.
- After Conv2D layer always used MaxPooling2D to downsample the spatial dimensions of the feature maps. It helps reduce the computational complexity of the model and extract the most important features from the input. With pooling size of (2, 2) with a stride of 2, the spatial dimensions of the feature maps are reduced by half.
- In almost all layers used activation function ReLU(simple non-linear function that takes the maximum between the input value and 0). Only for last Dense layer used Softmax activation function because it takes a vector of arbitrary real-valued scores and transforms them into a probability distribution over the classes. It outputs a vector of values between 0 and 1.
- For prevent overfitting used BatchNormalization(normalizes the activations of each layer) and Dropout(Randomly drops out a certain proportion of neurons). Dropout

used in the end with 0.5 which means 0.5 means that 50% of the input units will be randomly dropped out or deactivated.
- Flatten is using for prepare output of previous layers for Dense that is fully connected layer with X units(in our case is 512 and 1024 units)

Compiling model with specified parameters:

- Adam optimizer(most popular): Computes adaptive learning rates for each parameter. It is a stochastic gradient descent optimization algorithm.
- Sparse categorical cross-entropy loss function: Calculates the difference between the predicted probabilities and the true class label. Uses integers to represent the target class.

Callbacks using for perform certain actions at specific points during training, such as after each epoch or when a certain threshold is reached for a particular metric:

- EarlyStopping: callback that monitors a specified metric(in this case, validation loss and accuracy) and stops the training if the monitored metric stops improving.
- ReduceLROnPlateau: callback that reduces the learning rate by a factor(0.5) if the monitored metric does not improve for a specified number of epochs.
- TensorBoard: Log training and validation metrics to a TensorBoard file for visualization and analysis. Can be viewed in web UI

Train model with different approaches of data, first is simple that was loaded initially and second is using data augmentation:

- Steps per epoch is data size divided to batch size(which is 128, define how much images the model will process at a time during each training iteration)
- In first training process data was just loaded from cifar10 and used train for train and test for validation. After some 10 epochs model can be overfitted and because of this was used different approach for training in second process.
- In second training process. The augmented data is then used to train the model instead of the original data, which helps to increase the diversity of the data and prevent overfitting.

Results

After trainings and executing methods, the best result is Loss: 0.3-0.4, Accuracy: >0.9. After some epochs mostly in first training process validation parameters doesn't improves, which means model overfitting, because of this using data augmentation for second training. Also helps EarlyStopping's and ReduceLROnPlateau for reducing the learning rate.

If image that will be predicted is simple and can be for 100 % be clarified that this is this class, then model predict it with accuracy more than 0.9, but some hard images of birds that looks like airplanes model can't predict normally. Iterating over 25 images of the test data and predict each, accuracy was 100%