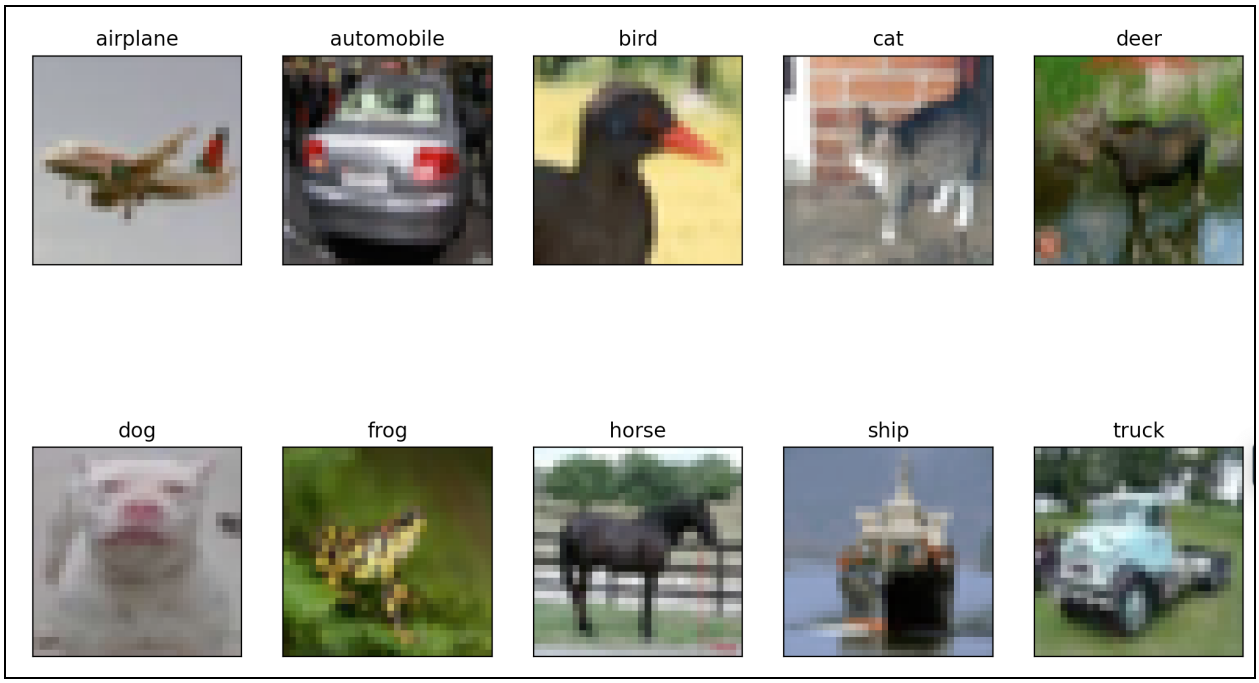
Aim

1. Using the code given in the lesson for classifying the CIFAR dataset, improve the quality of the model based on convolutional neural networks.
2. Make a report on model tuning, give a combination of parameters that provide the best quality indicators.
3. Provide graphs of changes in the parameters of the learning algorithm.
4. Describe one of the popular architectures for working with images

Dataset

The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Some examples:



Data preparing

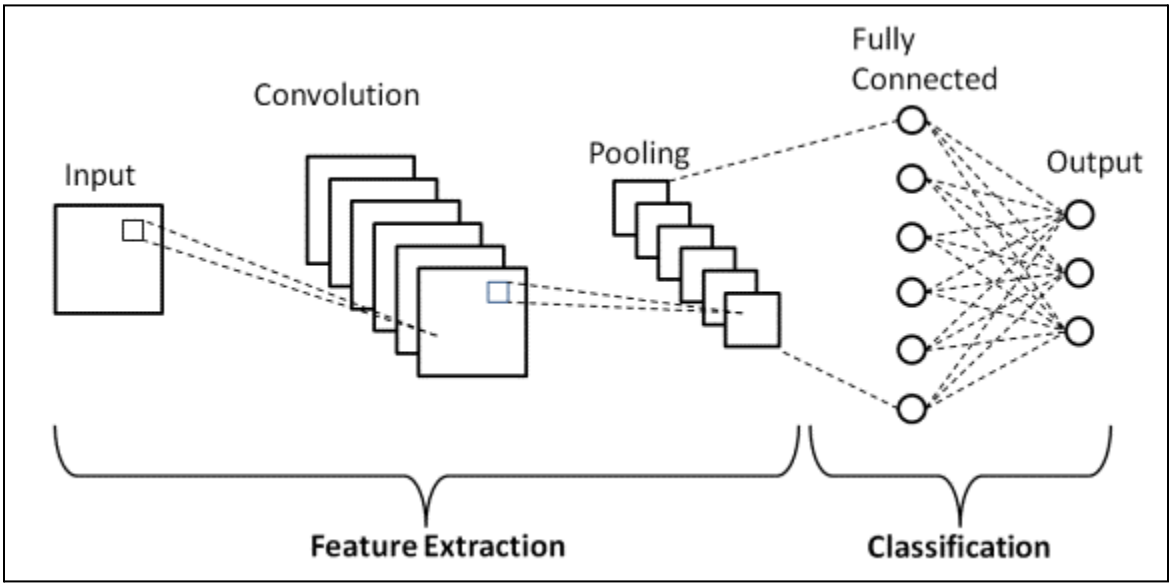
Everything is trivial here. We make categorical arrays from our data labels, so now every desired output is one-hot encoded. For better performance we also translate out inputs to a float type and normalize them by dividing by 255.

```
# Convert class vectors to binary class matrices.
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

# Normalizing the input image
x_train /= 255
x_test /= 255
```

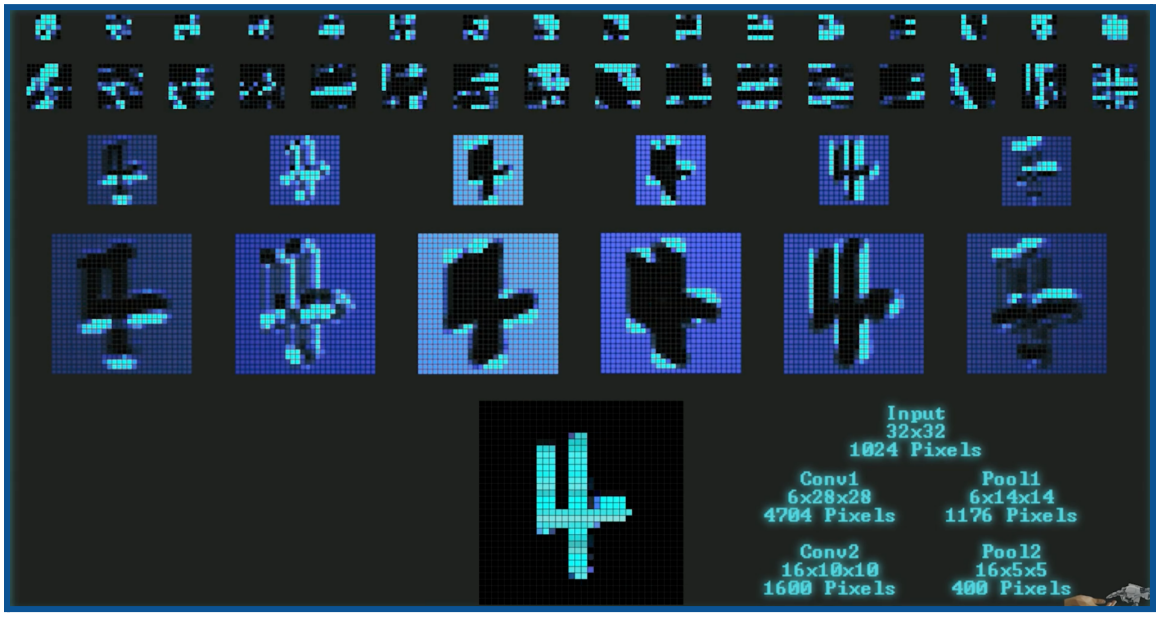
CNN basic architecture

CNNs are a class of Deep Neural Networks that can recognize and classify particular features from images and are widely used for analysing visual images.
Main parts of a CNN architecture are:

- A convolution tool that separates and identifies the various features of the image for analysis in a process called Feature Extraction.
- The network of feature extraction consists of many pairs of convolutional or pooling layers.
- A fully connected layer that utilizes the output from the convolution process and predicts the class of the image based on the features extracted in previous stages.
- This CNN model of feature extraction aims to reduce the number of features present in a dataset. It creates new features which summarises the existing features contained in an original set of features



An example of feature extraction with filtering and pooling can be observed in the following picture.

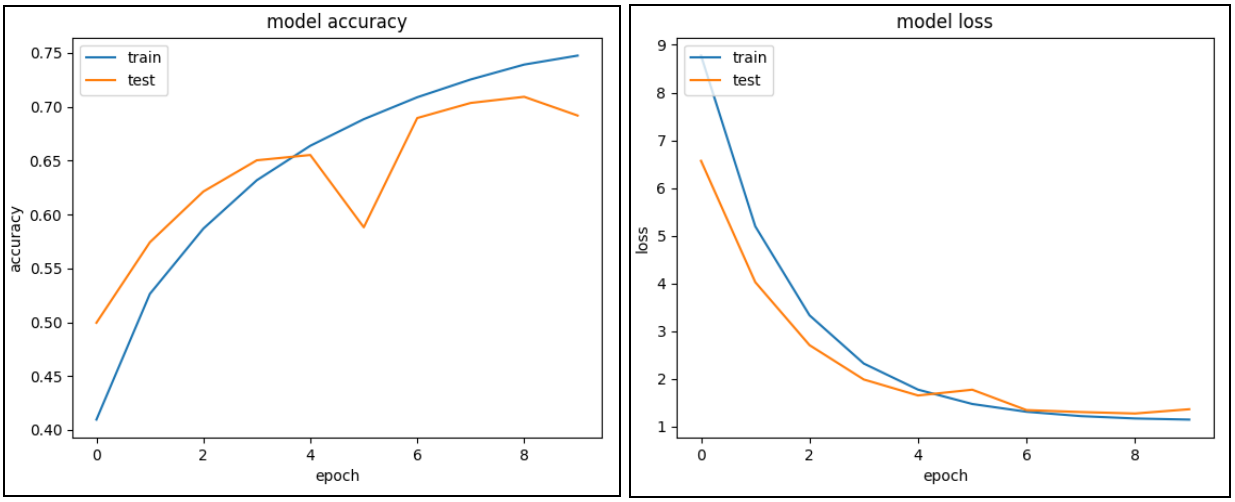


Model

With the help of TensorFlow ML framework we can implement our convolutional neural network object.
Here a sequential one is used. First data extraction step takes our data of shape 32x32x3 which is a number of one image pixels multiplied by the amount of channels, in our case is 3: red, green, blue. Sixty four filters will be created - so in the beginning 64 features of the picture will be extracted. Then, “Max Pooling”, which basically reduces the size of data (amount of pixels) by choosing the highest value of every four, will be used. This process is repeated twice (but with 128 filters in the second time), after which neuron signals are forwarded to a fully connected layer and with classical NN approach. The output is 10 classes (ten types of objects of the dataset). Dropout of 0.25 is applied in the whole model. Learning rate is 0.01.

```
model = Sequential()
model.add(Conv2D(64, (3, 3),
                 padding='same',
                 activation='relu',
                 input_shape=x_train.shape[1:]))

model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))


model.add(Conv2D(128, (3, 3),  activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512, kernel_regularizer=l2(0.01)))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

# compile
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
```

Firstly let’s try to evaluate our model with initial parameters which are discussed above. We have 10 epochs for that.



As we can see, validation accuracies are not as stable as train ones. The final evaluation gave us about 75% correct predictions on the train and ~69% on test data. An anomaly can be observed on the 6th epochs, where the performance dropped by 0.07. The whole model losses can be described with gradual decrease.

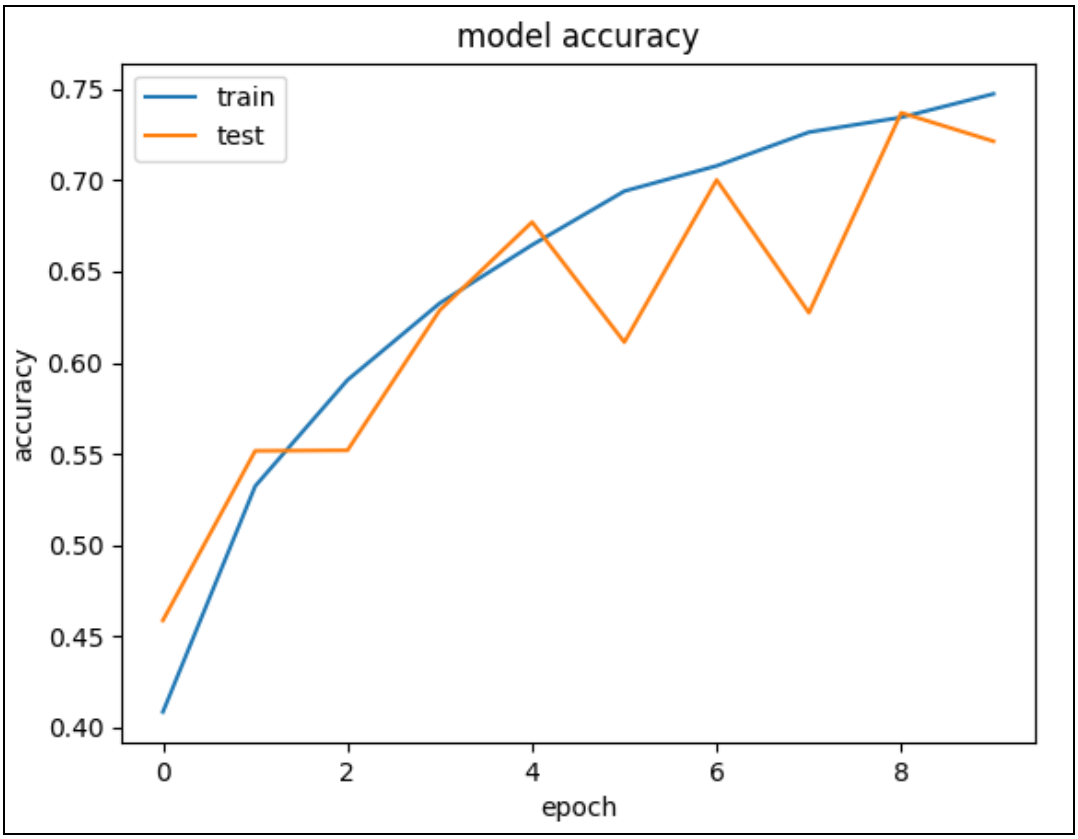
To alleviate our following hyper parameters tune, Optuna library will be used. Variables here are the number of filters on convolutional layers, pooling size, dropout, learning rate and amount of nodes on fully connected dense layer. All the code needed:

```
def objective(trial):
    params = {
        'learning_rate': trial.suggest_float('learning_rate', 0.001, 0.1),
        'dropout': trial.suggest_float('dropout', 0.0, 0.4),
        'conv1_num': trial.suggest_int('conv1_num', 10, 100),
        'conv2_num': trial.suggest_int('conv2_num', 40, 200),
        'fcl_num': trial.suggest_int('fcl_num', 200, 700),
        'max_pooling': trial.suggest_int('max_pooling', 2, 4)
    }

    model, history = create_model(
        conv1_num=params['conv1_num'],
        conv2_num=params['conv2_num'],
        fcl_num=params['fcl_num'],
        max_pooling=params['max_pooling'],
        learning_rate=params['learning_rate'],
        dropout=params['dropout']
    )
    return history.history['accuracy'][-1]

study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=30)
```

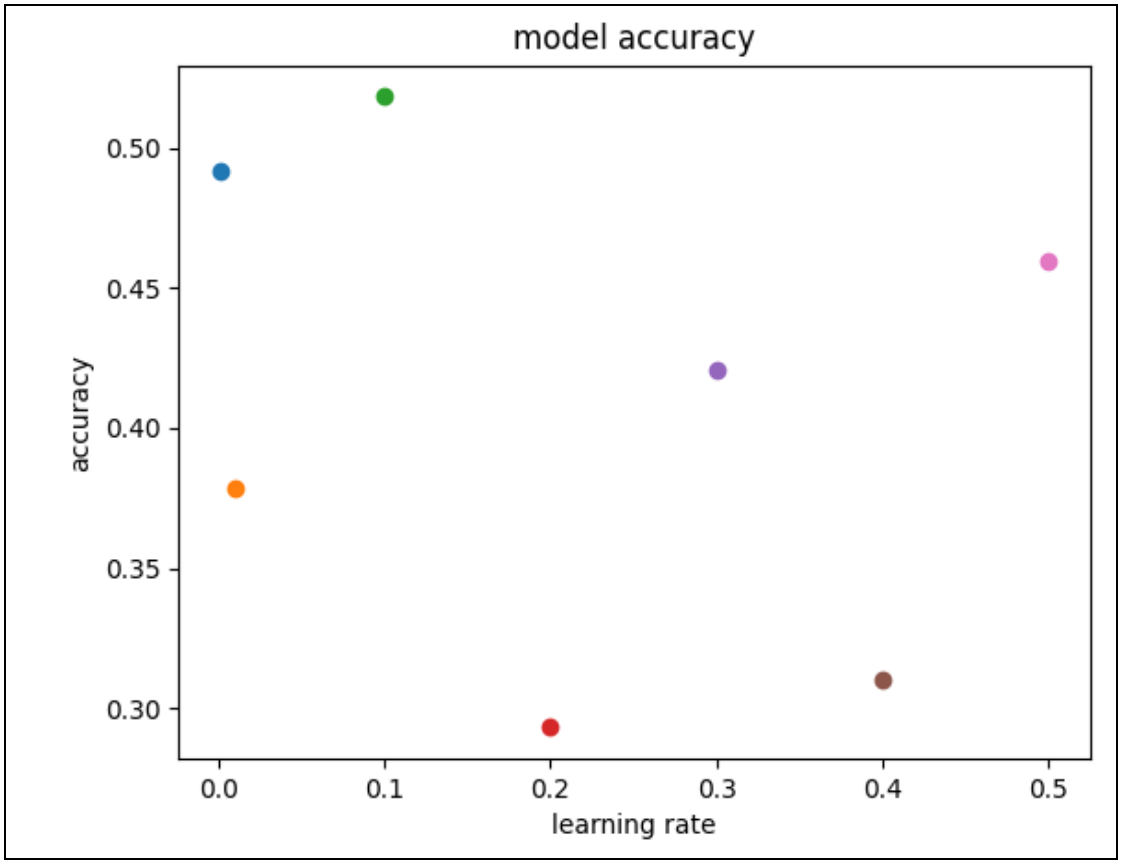
After multiple studies we can state that desirable parameters can be next:

learning_rate: 0.07,
dropout: 0.02,
conv1_num: 100,
conv2_num: 95,
fcl_num: 568,
max_pooling: 2

We train our model on those and get the same result as before but it looks like the model was overfitting while training and test evaluation performed higher by ~2%. Seeing a general trend it can be anticipated that more epochs could lead to higher accuracy.



After a try of tuning the learning rate individually we faced irregularity and very chaotic behaviour of our model. Anyways the best parameters of learning rate are 0.01 and 0.1.



To sum it can be stated that convolutional type of neural networks is the best one for the large image (pixels amount) classification, the principal difference from perceptron for example is a data extraction in the beginning, which reduces the size of input arrays - so it takes less time and memory for the computer to do some calculations.