# PITE Report

Pampuch Team

## Problem description

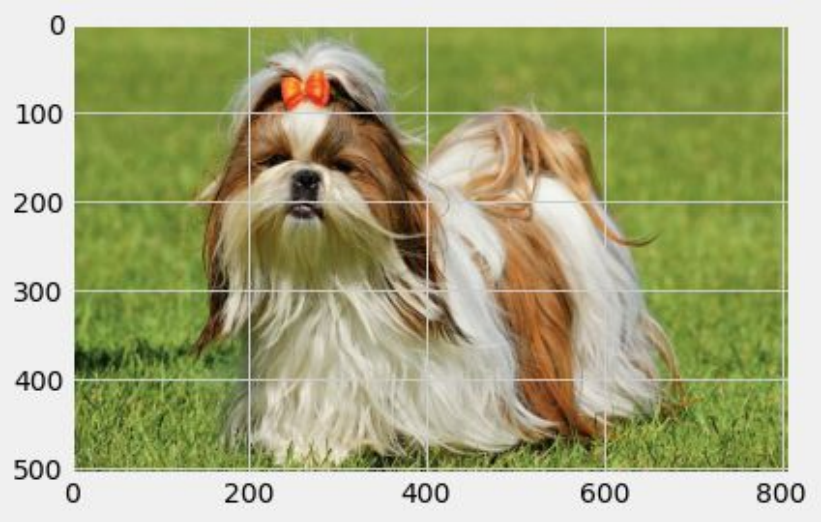
The problem is object categorization with machine learning model and object detection with implement YOLO algorithm. Both of those models will be used as a backend to web application.

## How does it work

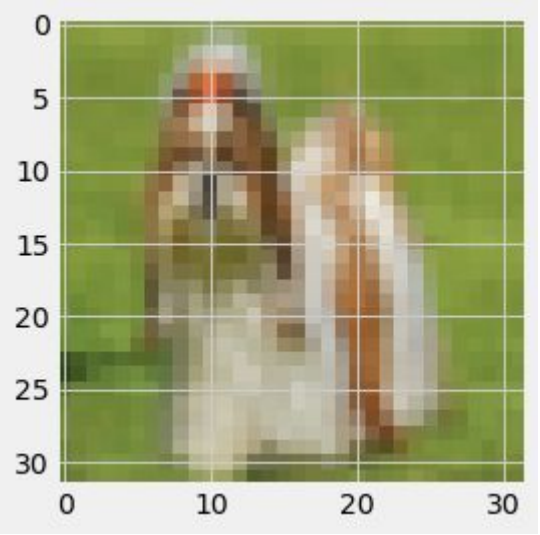
Image categorization is part of machine learning and neural networks are most commonly used to solve this problem. The most popular one are convolutional neural networks and recurrent neural networks. We choose to go with convolutional neural network.

The model takes an image as an input and returns the predictions with probabilities of object on an image.

Eq. Original photo:




Which are then scaled to required dimensions (in our case it's 32x32):



The algorithm returns predictions with probabilities:

dog : 46.73 %
bird : 19.45 %
cat : 11.8 %
deer : 11.08 %
horse : 6.73 %

Top 5 predictions for the image above

```
model.add(Conv2D(32, (5, 5), activation='relu', input_shape=(32,32,3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(1000, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(500, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(250, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

Our algorithm is composed of 13 layers.

The first layer is a convolutional 2D layer. Which as an input takes $input\_shape = (32, 32, 3)$ for 32x32 RGB pictures. It performing a matrix multiplication operation between selected proportion of selected matrix in our case it's $(5, 5)$ and the portion of the image over which the kernel is hovering doing it for every input channel then adding the result for the final matrix.

The second layer is Pooling.
"Downsamples the input representation by taking the maximum value over the window defined by $pool\_size$ for each dimension along the features axis."[1]
There are two more sets of Conv2D and MaxPolling2D layers in order to scale and analize the image further down.

Next layer Flatten flattens the input. Flattening a tensor means to remove all of the dimensions except for one.[2] This changed the input which is an arrays of matrixes to array of neurons

Next layer is the Dense layer.
"Dense implements the operation: output = activation(dot(input, kernel) + bias) where activation is the element-wise activation function passed as the activation argument,

[1] "MaxPooling2D layer - Keras." https://keras.io/api/layers/pooling_layers/max_pooling2d.
[2] "How does the Flatten layer work in Keras? - Stack Overflow." 26 May. 2017, https://stackoverflow.com/questions/44176982/how-does-the-flatten-layer-work-in-keras.

kernel is a weights matrix created by the layer, and bias is a bias vector created by the layer (only applicable if use_bias is True).”[3]

Next layer Dropout prevents overfitting by setting input units to 0 with frequency passed as an argument.
The repetition of layers serves to scale down the amount of neurons to the size of our labels. In our case we are using cifar10 so there are 10 labels.

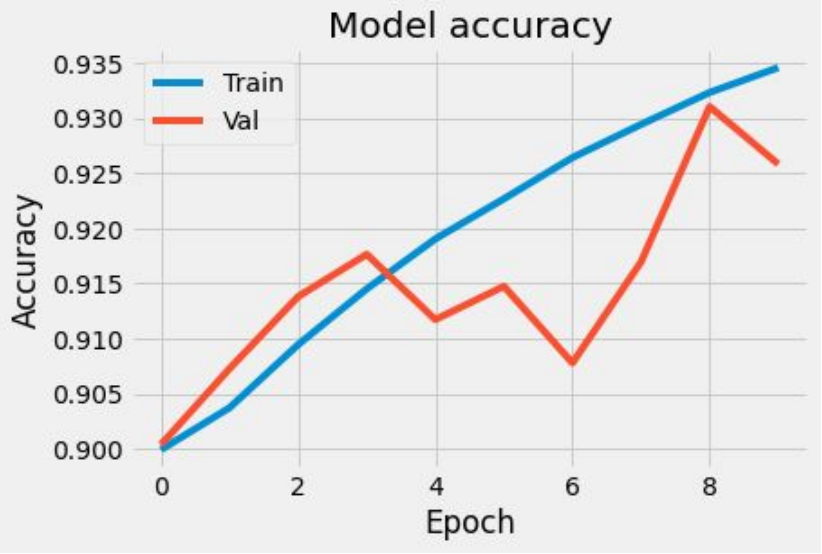
## Algorithm evolution

https://docs.google.com/document/d/1cWIkea1KiFxrd3m0WJ55s3r3S70rYHm9KBPgaZRxplA/edit?usp=sharing
Following on from the previous report. We experimented with model by adding additional neural network layers.

```
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

It allowed us to further analyze the image even more. This change helped tremendously with the model's accuracy. It went up to 86%. Adding additional convolutional layers and making the check matrix bigger. Didn’t help at all. Model was now more overfitted. Checking on random internet photos showed worse results.
Between every change we experimented with a number of epochs. Which didn’t affect the accuracy of the model at all.
Next we started experimenting with loss function and optimizer. We checked the $binary\_crossentropy$ as new loss function and optimizer from $Adam$ to $rmsprop$. Which increased the accuracy to around 93%.




[3] "Dense layer - Keras." https://keras.io/api/layers/core_layers/dense.

As we set up our objective to have accuracy to be at least 90 %. We only experimented with a number of epochs up to 30, but this didn't increase accuracy. We went back to 10 epochs cause it didn't change accuracy at all and we were scared of overfitting model

## YOLO - How does it work

Yolo - You only look once is an object detection algorithm. We are using the third iteration named YoloV3[4]. YOLO uses a totally different approach. We apply a single neural network to the full image. This network divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities. The most salient feature of this generation is detection at three different sizes. It makes predictions at three different scales in order to better detect different size objects. "In YOLO v3, the detection is done by applying 1 x 1 detection kernels on feature maps of three different sizes at three different places in the network."[5]

[4] "qqwweee/keras-yolo3 - GitHub." https://github.com/qqwweee/keras-yolo3.
[5] "What's new in YOLO v3? - Towards Data Science." 23 Apr. 2018, https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b.