Report

Neural networks and deep learning

Subject:

Image classification using convolutional neural networks

Piotr Jaśkowski

1.1 Project goals

The goal of the project was to create a model capable of differencing low resolution images of objects.

1.2 Technologies

matplotlib - is a graphing library for the Python programming language and its NumPy extension.

numpy - is a library that supports large, multidimensional arrays and matrices, along with a large collection of high-level mathematical functions for operating on these arrays.

TensorFlow - is an open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

2.1 Data

The data that I'm using for this project is taken tensorflows website. It's a collection of 60 000 images with various objects, labeled over 10 categories. The categories are: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck. The images are 32x32 and they are colored.

2.1 Data preparation

The dataset shape looks like this:

```
[3] print(x_train.shape)
    print(y_train.shape)

    (50000, 32, 32, 3)
    (50000, 1)
```

The image shape 32, 32, 3 shows that images are colored which is not preferable as models made for colored images need bigger inputs. In the following loops the images have been transformed to black and white spectrum to reduce input data.

```
[4] new_x_train = []
    new_x_test = []

    for image in x_train:
      new_x_train.append(cv2.cvtColor(image, cv2.COLOR_BGR2GRAY))

    for image in x_test:
      new_x_test.append(cv2.cvtColor(image, cv2.COLOR_BGR2GRAY))

    x_train = np.array(new_x_train)
    x_test = np.array(new_x_test)
```

After this change data has been reshaped.

```
x_train = x_train.reshape(50000, 32, 32)
x_test = x_test.reshape(10000, 32, 32)
```

```
y_train = y_train.reshape((50000, ))
y_test = y_test.reshape((10000, ))
```

Range of grayscale has been changed from 0 - 255 to 0.0 - 1.0 as to normalize data and to make imputs smaller.

```
[160] x_train = (x_train - np.min(x_train))/np.ptp(x_train)
      x_test = (x_test - np.min(x_test))/np.ptp(x_test)
```

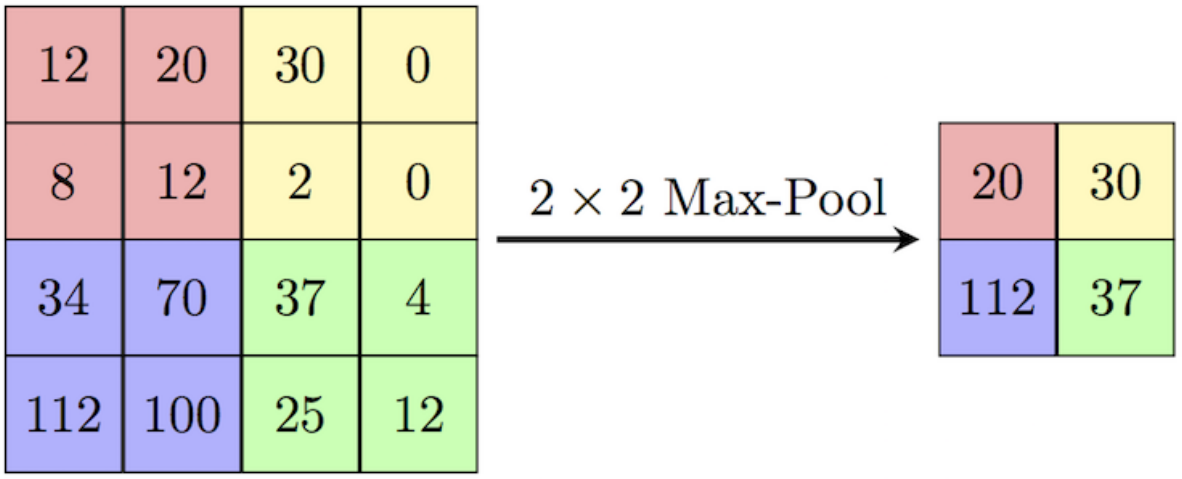
After that the dataset is ready for neural networks.

3.1 Modeling

The first model is very similar to the one form laboratory 9 as it is in the same subject of image classification. It's A simple model used by me to benchmark data and go through all the bassics.

```
model = Sequential()
layer_1 = layers.Conv2D(64, kernel_size=(2,2), padding="same", activation = "relu", input_shape=(32,32,1))
layer_2 = layers.MaxPooling2D(pool_size=(2), strides=(2))
layer_3 = layers.Conv2D(32, kernel_size=(2,2), padding="same", activation = "relu", input_shape=(32,32,1))
layer_4 = layers.MaxPooling2D(pool_size=(2), strides=(2))
layer_5 = layers.Flatten()
layer_6 = layers.Dense(256, activation = "relu")
layer_7 = layers.Dense(10, activation = "softmax")
```

It is a sequential model with 7 layers, with 2d convolution layers as an input. The input shape is 32,32,1 as is the image shape. After both convolutional layers there is Max Pooling which is a method that calculates the maximum value for patches of a feature image, and uses it to create a downsampled (pooled) image map.



Activation functions are relu as I found them to work best for this type of problem. On the output layer there is a softmax activation function with 10 outputs one for each class. It is a very common output activation function for a multiclass probability.

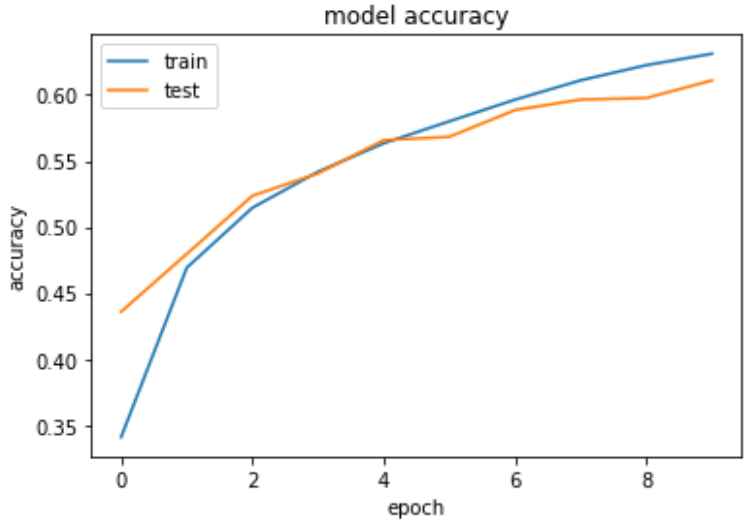
Rest of the models parameters:

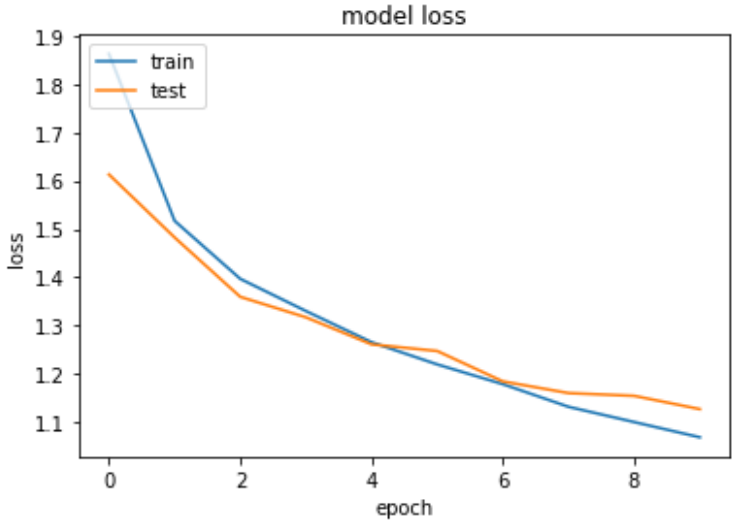
```
model.compile(
 optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3, ),
 loss='sparse_categorical_crossentropy',
 metrics=['accuracy'])

predictions = model.fit(
 x_train, y_train,
 epochs= 10,
 batch_size = 500,
 validation_data = (x_test, y_test)
)
```

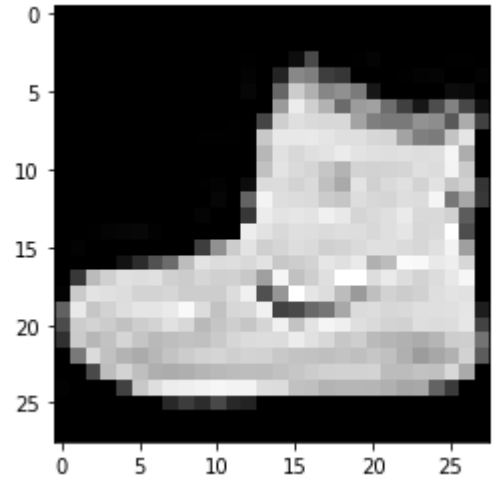
Model result:

```
Metrics:
Test-Accuracy: 0.5506999999284744
Test-Loss: 1.2910549521446228
```





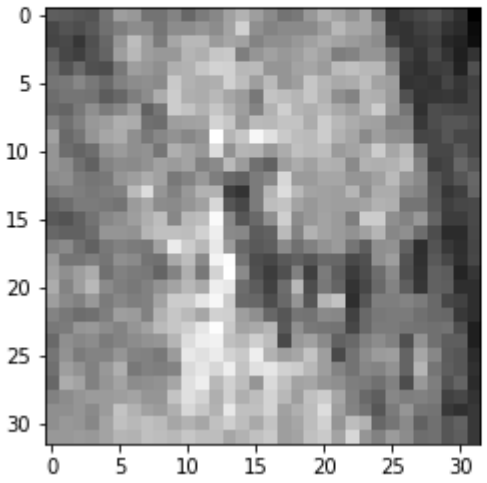
Generally the model is pretty bad with just 55% accuracy, a lot can be accounted to low quality images of complex objects with low contrast. Below just to illustrate how bad they are, first is a sample image from mnist dataset:



It's also a low resolution dataset howere objects are in high contrast and you can clearly recognize them, belowe sample image from my dataset:

```
[25] plt.imshow(x_train[10], cmap=plt.cm.gray)
```

```
<matplotlib.image.AxesImage at 0x7f13084c3150>
```



I think the difference here is obvious, the low quality of images is the fundamental reason for a mediocre accuracy score. The low quality of images is a principal challenge for this project.

3.1 Tweaking Models

To solve more complex problems more fine tuning of models was needed. With the help of plugin TensorBoard HParams lots of tests with different hyperparameters were done to find the best fits. Below list of values to try and log model configuration and accuracy score to TensorBoard.

```
[177] HP_NUM_UNITS = hp.HParam('num_units', hp.Discrete([16, 32, 64]))
      HP_NUM_UNITS_2 = hp.HParam('num_units_2', hp.Discrete([256, 512]))
      HP_DROPOUT = hp.HParam('dropout', hp.RealInterval(0.2, 0.5))
      HP_OPTIMIZER = hp.HParam('optimizer', hp.Discrete(['adam', 'sgd', 'rmsprop']))
      HP_L1L2 = hp.HParam('l1l2 regularizer', hp.RealInterval(.001,.01))
      HP_POOL_SIZE = hp.HParam('pool size', hp.Discrete([2, 4]))

      METRIC_ACCURACY = 'accuracy'

      with tf.summary.create_file_writer('logs/hparam_tuning2').as_default():
        hp.hparams_config(
          hparams=[HP_NUM_UNITS, HP_NUM_UNITS_2, HP_DROPOUT, HP_OPTIMIZER, HP_L1L2, HP_POOL_SIZE],
          metrics=[hp.Metric(METRIC_ACCURACY, display_name='Accuracy')],
        )
```

Below function creates new models with hparams as an input. The first input layer has a number of neurons defined by hpatams[HP_NUM_UNITS], those are hyperparameters seen in the screenshot above. Almost all of the layers have some hyperparameters. Parameters that are not changed are standard to the problem, that is for example relu activation function on input layer, softmax activation function on out layer or number of output neurons. Those things are defined by the nature of the problem, which is image classification. The output layer always has to have 10 out neurons, because there are 10 classes and so on.

```
[178] def train_test_model(hparams):
        model = tf.keras.models.Sequential([
          tf.keras.layers.Conv2D(hparams[HP_NUM_UNITS], kernel_size=(2,2),
                                 kernel_regularizer=tf.keras.regularizers.L1L2(hparams[HP_L1L2]),
                                 activation=tf.nn.relu, input_shape=(32,32,1)),
          tf.keras.layers.MaxPooling2D(pool_size=(hparams[HP_POOL_SIZE]), strides=(hparams[HP_POOL_SIZE])),
          tf.keras.layers.Dropout(hparams[HP_DROPOUT]),
          tf.keras.layers.Flatten(),
          tf.keras.layers.Dense(hparams[HP_NUM_UNITS_2], activation=tf.nn.relu),
          tf.keras.layers.Dense(10, activation=tf.nn.softmax),
        ])
        model.compile(
            optimizer=hparams[HP_OPTIMIZER],
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'],
        )

        model.fit(x_train, y_train, epochs=5, batch_size = 500)
        _, accuracy = model.evaluate(x_test, y_test)
        return accuracy
```

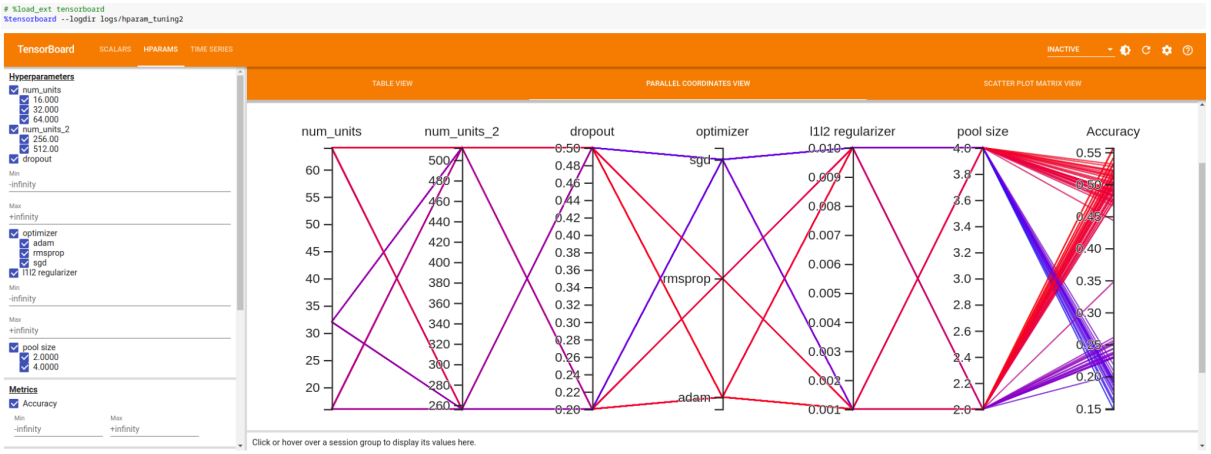
For each time a new model is created its parameters and accuracy are saved to a local file. For loops below are running all the possible variation of model with selected before hyperparameters.

```
def run(run_dir, hparams):
  with tf.summary.create_file_writer(run_dir).as_default():
    hp.hparams(hparams)  # record the values used in this trial
    accuracy = train_test_model(hparams)
    tf.summary.scalar(METRIC_ACCURACY, accuracy, step=1)
```

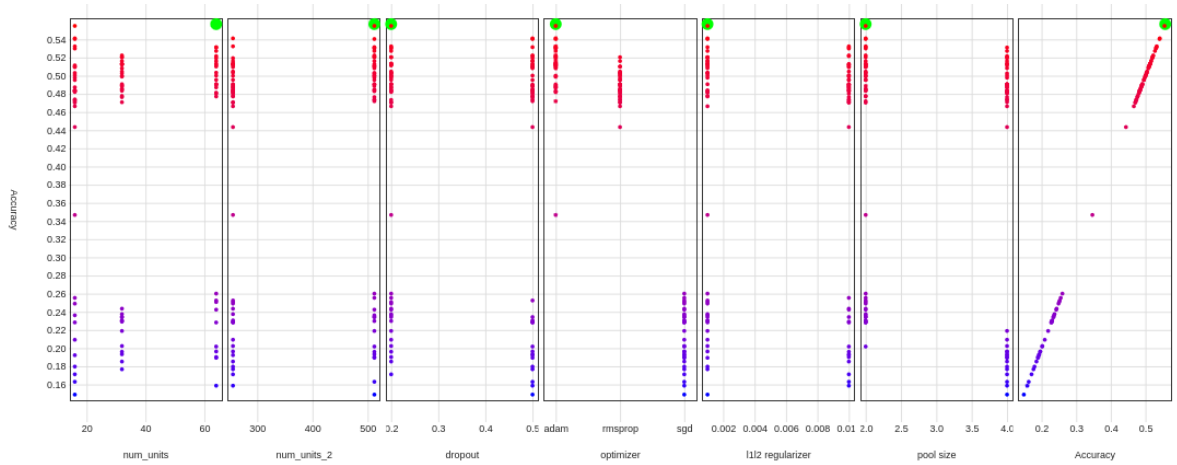
```
[ ] for num_units in HP_NUM_UNITS.domain.values:
      for num_units_2 in HP_NUM_UNITS_2.domain.values:
        for dropout_rate in (HP_DROPOUT.domain.min_value, HP_DROPOUT.domain.max_value):
          for optimizer in HP_OPTIMIZER.domain.values:
            for L1L2 in (HP_L1L2.domain.min_value, HP_L1L2.domain.max_value):
              for pool_size in HP_POOL_SIZE.domain.values:
                hparams = {
                    HP_NUM_UNITS: num_units,
                    HP_NUM_UNITS_2: num_units_2,
                    HP_DROPOUT: dropout_rate,
                    HP_OPTIMIZER: optimizer,
                    HP_L1L2: L1L2,
                    HP_POOL_SIZE: pool_size,
                }
                run_name = "run-%d" % session_num
                print('--- Starting trial: %s' % run_name)
                print({h.name: hparams[h] for h in hparams})
                run('logs/hparam_tuning2/' + run_name, hparams)
```

Running through all the variations of models takes some time but afterwards a tensorboard plugin raport is generated. Below is a representation of all the variation, colored red means accuracy is high and the purple means accuracy is low. The most obvious bad parameter is optimizer ‘sgd’, as all the connections to nodes representing it are purple. On closer inspection more find tuned differences can be observed. Lower regulizers are better for accuracy, ‘adam’ optimizer is

better then ‘rmsprop’, more neurons in second layer also improves accuracy.



On scatter plots below we can see with green selected the best combination of hyperparameters. With High number of input neurons, high number of neurons on the second layer, low dropout rate and low regulizers. Here we also see that ‘rmsprop’ is worse than ‘adams’ and that medium size of input neurons is performing worse then low number or high.

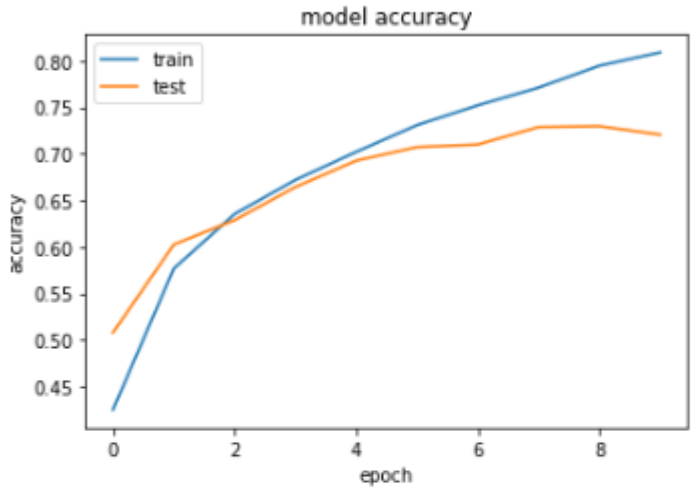


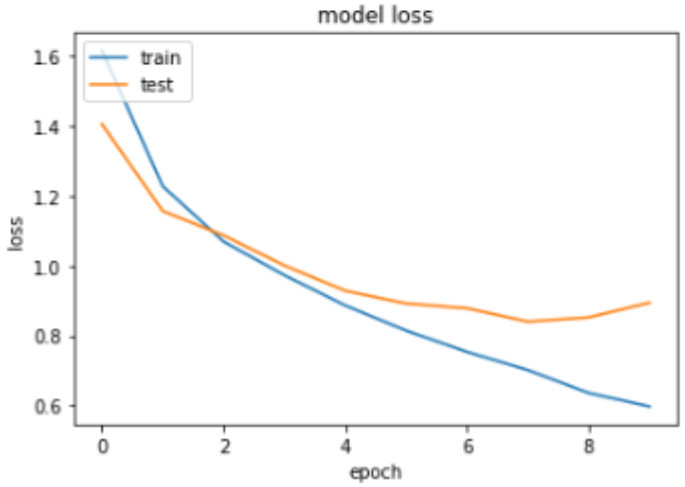
With all that new information a new model was created with a higher number of neurons, lower regulizers and lower dropout. All of the regular expected options are here, Conv2D layer, relu activator, adam optimizer and so on.

```
model = Sequential()
layer_1 = layers.Conv2D(64, kernel_size=(2,2), padding="same",
                        activation = "relu", input_shape=(32,32,1),
                        kernel_regularizer=regularizers.l2(0.0001))
layer_2 = layers.MaxPooling2D(pool_size=(2), strides=(2))
layer_3 = layers.Conv2D(128, kernel_size=(2,2), padding="same",
                        activation = "relu", input_shape=(32,32,1),
                        kernel_regularizer=regularizers.l2(0.0001))
layer_4 = layers.MaxPooling2D(pool_size=(2), strides=(2))
layer_5 = layers.Conv2D(256, kernel_size=(2,2), padding="same",
                        activation = "relu", input_shape=(32,32,1),
                        kernel_regularizer=regularizers.l2(0.0001))
layer_6 = layers.MaxPooling2D(pool_size=(2), strides=(2))
layer_7 = layers.Dropout(0.2)
layer_8 = layers.Flatten()
layer_9 = layers.Dense(512, activation = "relu")
layer_10 = layers.Dense(10, activation = "softmax")
```

Accuracy is 67% which is an improvement from 55% in previous mode, Loss is also lower in the new model 0.99 vs 1.29 in the old one. On plot we can see test accuracy flatten after epoch 6 and even slight decrease after epoch 8, any further training of this model will only over-fit it.

```
Metrics:
Test-Accuracy: 0.669130003452301
Test-Loss: 0.9933655619621277
```





Summary

As project two models were created with a purpose of image classification, first one was basic image classification model, second was tweaked many times to achieve best score. Improvements are noticeable however the end result is still not great, I attribute it to challenging dataset with low resolution, and low contrast.