Deep Learning and Reinforcement Learning

IBM Skills Network

Project Report

By:

Fahd Seddik

Table of Contents

Main Objective

Image classification has been one of the most demanded application for deep learning. Many people have tried different deep learning approaches and neural network architectures in order to come up with the best way to solve image classification problems.

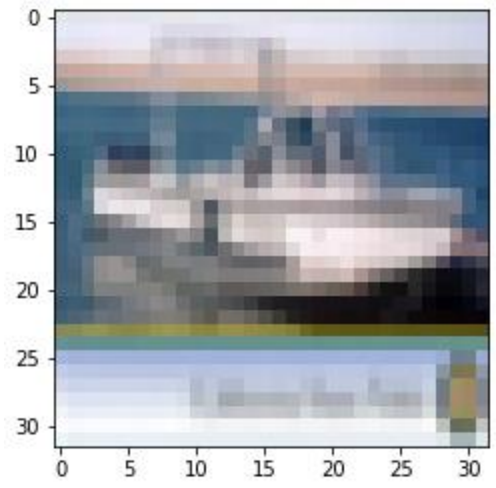
In this project we will be trying to distinguish between a set of 10 classes using an image dataset as we will see in the next section.

Brief Description

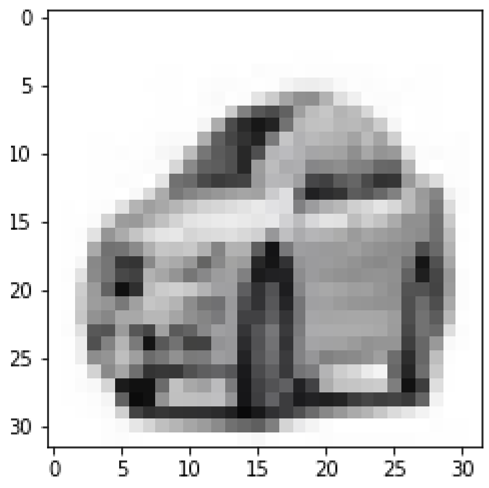
We will work with the CIFAR-10 Dataset. This is a well-known dataset for image classification which consists of **60000** 32x32 color images each of which are one of 10 classes with 6000 images per class. As shown below, these are some of the images in our dataset along with their labels. The 10 classes we need to predict are as follows:

0. Airplane
1. Automobile
2. Bird
3. Cat
4. Deer
5. Dog
6. Frog
7. Horse
8. Ship
9. Truck

SHIP

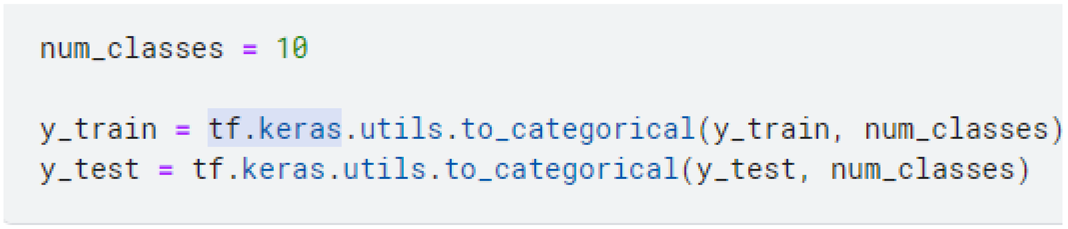


CAR



Data cleaning & Feature engineering

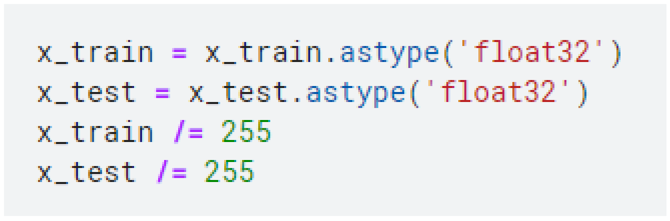
First, we retrieve our train and test sets split into X_train, y_train, X_test, and y_test. However, we need to transform each of our target variables into the needed format by transforming them into categorical values. This will convert our data into a vector of all zeroes except the needed column.



```
num_classes = 10

y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)
```

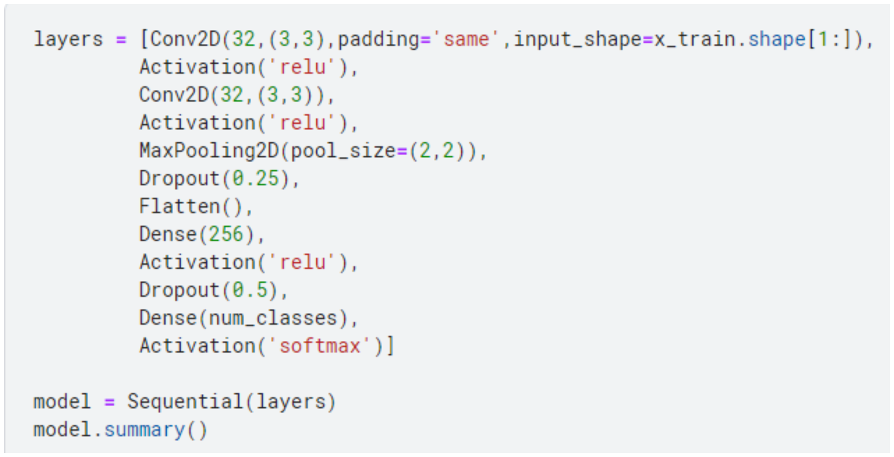
After that, we will be needed to make everything float values and scale them down to be from 0 to 1. This will be done by dividing each of the values in our x sets by 255 since each value is between 0 and 255.



```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
```

Deep Learning Models

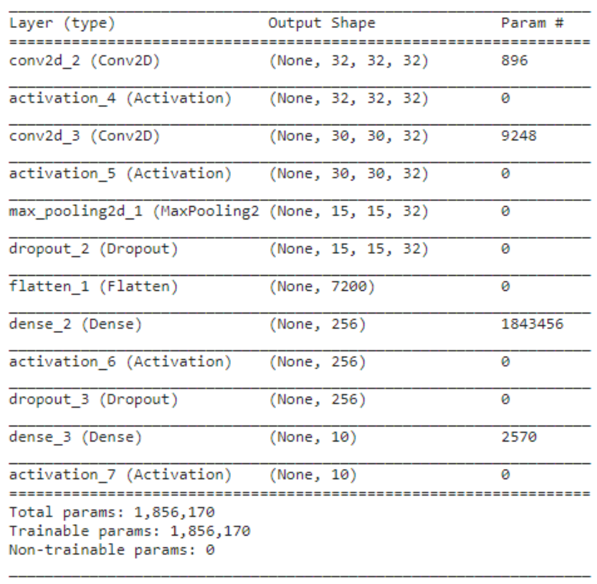
We will be using 3 different architectures for **Convolutional Neural Networks** to try to find the optimum architecture out of all of them that best suits the given classification problem. The first architecture would consist of 2 convolutional layers followed by a Max-Pooling layer and a dropout layer. After that we will flatten and import that to our next network. The network would be made up of a dense layer, drop out layer, and another dense layer at the end which would output the needed class with SoftMax activation function.



```
layers = [Conv2D(32,(3,3),padding='same',input_shape=x_train.shape[1:]),
          Activation('relu'),
          Conv2D(32,(3,3)),
          Activation('relu'),
          MaxPooling2D(pool_size=(2,2)),
          Dropout(0.25),
          Flatten(),
          Dense(256),
          Activation('relu'),
          Dropout(0.5),
          Dense(num_classes),
          Activation('softmax')]

model = Sequential(layers)
model.summary()
```

First Architecture params.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
activation_4 (Activation)    (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 30, 30, 32)        9248
_________________________________________________________________
activation_5 (Activation)    (None, 30, 30, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 15, 15, 32)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 15, 15, 32)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 7200)              0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               1843456
_________________________________________________________________
activation_6 (Activation)    (None, 256)               0
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                2570
_________________________________________________________________
activation_7 (Activation)    (None, 10)                0
=================================================================
Total params: 1,856,170
Trainable params: 1,856,170
Non-trainable params: 0
_________________________________________________________________
```

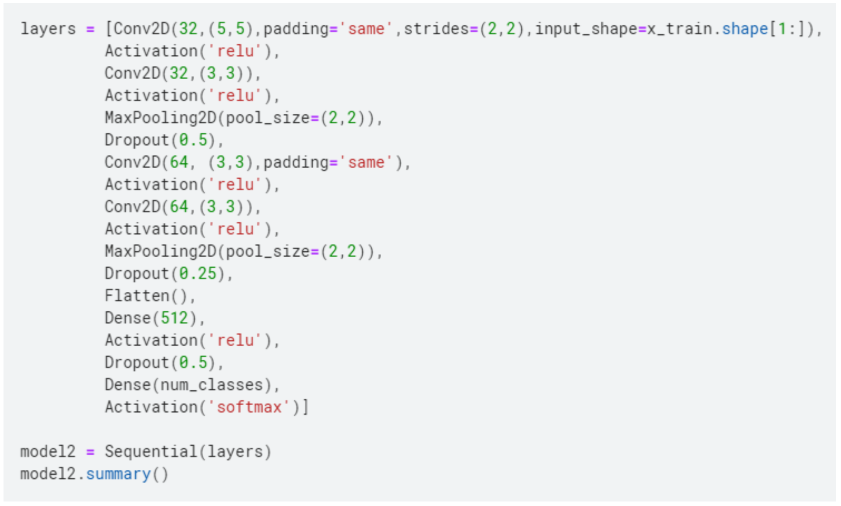
As you can see from the figure shown here, we will be needing to train approx. 1.86 million parameters during our training process for our first model.

We will be using a categorical cross-entropy function as our loss function, an RMSprop optimizer, and an accuracy metric. For the fitting process, the model would be trained with batch size of 32 and for 15 epochs using shuffling.

The model's results were having a final validation accuracy of 64.5% and loss of 0.976. The model's training time for each epoch was around 1 minute, summing up to 15 minutes for the 15 epochs we made. This would be used later on in order to compare it to other models.

As for the second architecture we would increase the number of convolutional layers to see if we can capture more of the features while also reducing the number of parameters needed in each layer trying to be closer to a VGG architecture but not quite.

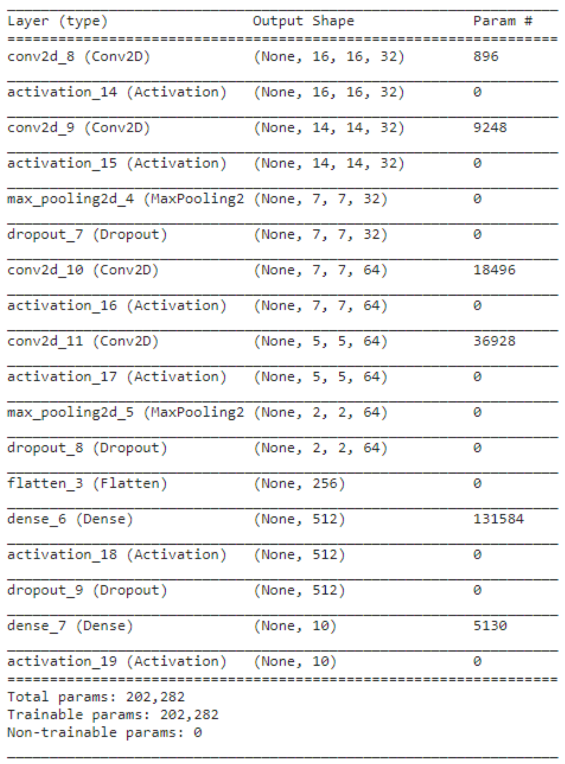
As you can see, this architecture has much less parameters that need to be trained compared to the previous model. This would only need to train 202K parameters. As for our optimizer, loss function and metric, we will be making them constant throughout our different architectures to make sure we have a correct comparison that is not biased. We will also train this model using 15 epochs.



```
layers = [Conv2D(32,(5,5),padding='same',strides=(2,2),input_shape=x_train.shape[1:]),
          Activation('relu'),
          Conv2D(32,(3,3)),
          Activation('relu'),
          MaxPooling2D(pool_size=(2,2)),
          Dropout(0.5),
          Conv2D(64, (3,3),padding='same'),
          Activation('relu'),
          Conv2D(64,(3,3)),
          Activation('relu'),
          MaxPooling2D(pool_size=(2,2)),
          Dropout(0.25),
          Flatten(),
          Dense(512),
          Activation('relu'),
          Dropout(0.5),
          Dense(num_classes),
          Activation('softmax')]

model2 = Sequential(layers)
model2.summary()
```

Second Architecture params.

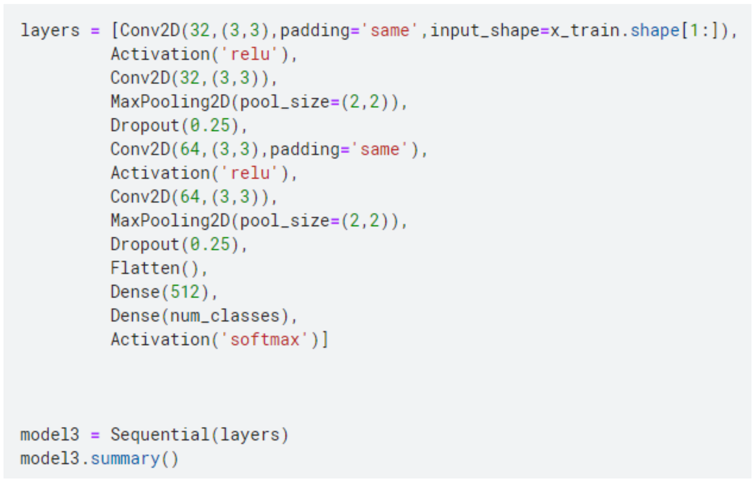


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)            (None, 16, 16, 32)        896
_________________________________________________________________
activation_14 (Activation)   (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 14, 14, 32)        9248
_________________________________________________________________
activation_15 (Activation)   (None, 14, 14, 32)        0
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 32)          0
_________________________________________________________________
dropout_7 (Dropout)          (None, 7, 7, 32)          0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 7, 7, 64)          18496
_________________________________________________________________
activation_16 (Activation)   (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 5, 5, 64)          36928
_________________________________________________________________
activation_17 (Activation)   (None, 5, 5, 64)          0
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 2, 2, 64)          0
_________________________________________________________________
dropout_8 (Dropout)          (None, 2, 2, 64)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 256)               0
_________________________________________________________________
dense_6 (Dense)              (None, 512)               131584
_________________________________________________________________
activation_18 (Activation)   (None, 512)               0
_________________________________________________________________
dropout_9 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_7 (Dense)              (None, 10)                5130
_________________________________________________________________
activation_19 (Activation)   (None, 10)                0
=================================================================
Total params: 202,282
Trainable params: 202,282
Non-trainable params: 0
_________________________________________________________________
```

The model's results were having a final validation accuracy of 67.7% and loss of 1.0803. The model's training time for each epoch was around 30 seconds, summing up to around 7.5 minutes for the 15 epochs we made. This would be used later on in order to compare it to other models.

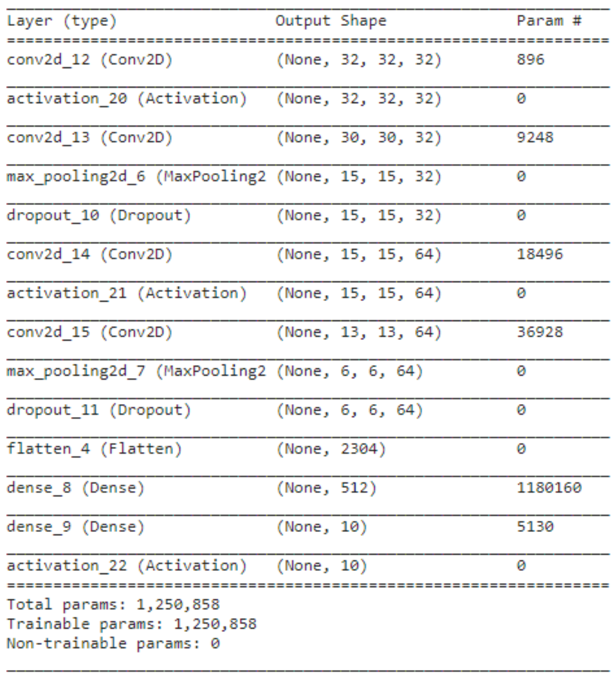
The last architecture will have an aim of getting the best out of both models. It would many parameters but not so much as our first model and not so few as our second model. This would try to have an adequate training time while still trying to get the best accuracy it can. As you can see below, the model has 1.25 million parameters to train.

Third Architecture params.



```
layers = [Conv2D(32,(3,3),padding='same',input_shape=x_train.shape[1:]),
          Activation('relu'),
          Conv2D(32,(3,3)),
          MaxPooling2D(pool_size=(2,2)),
          Dropout(0.25),
          Conv2D(64,(3,3),padding='same'),
          Activation('relu'),
          Conv2D(64,(3,3)),
          MaxPooling2D(pool_size=(2,2)),
          Dropout(0.25),
          Flatten(),
          Dense(512),
          Dense(num_classes),
          Activation('softmax')]


model3 = Sequential(layers)
model3.summary()
```



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 32, 32, 32)        896
_________________________________________________________________
activation_20 (Activation)   (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 30, 30, 32)        9248
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 15, 15, 32)        0
_________________________________________________________________
dropout_10 (Dropout)         (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 15, 15, 64)        18496
_________________________________________________________________
activation_21 (Activation)   (None, 15, 15, 64)        0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 13, 13, 64)        36928
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
dropout_11 (Dropout)         (None, 6, 6, 64)          0
_________________________________________________________________
flatten_4 (Flatten)          (None, 2304)              0
_________________________________________________________________
dense_8 (Dense)              (None, 512)               1180160
_________________________________________________________________
dense_9 (Dense)              (None, 10)                5130
_________________________________________________________________
activation_22 (Activation)   (None, 10)                0
=================================================================
Total params: 1,250,858
Trainable params: 1,250,858
Non-trainable params: 0
_________________________________________________________________
```

For the same reason as before, we will be using same optimizer, loss function, and metric to train our model on 15 epochs with same batch size. The model's results were having a final validation accuracy of 73.76% and loss of 0.6186. The model's training time for each epoch was around 1.5 minutes, summing up to around 22.5 minutes for the 15 epochs we made. In the next section, we will be stating which of the models is recommended.

Recommended Model

Given the accuracies of each of the models, the training time, and the number of parameters needed to be trained (i.e. computational complexity). We can say that the **Third architecture** is the **best** out of all of them. This is due to it having a much better accuracy than the other models and even though it took more time to train but the difference in accuracy is substantial and worth the extra time for training. This would be much more practical if we need to further train the model for enhancing its results.

Key Findings and Insights

We can obviously see that the first model took a lot of time to train but did not do as well as the second even though it was much more complex. However, this is probably due to the fact the it has more parameters to train which made it more likely to over-fit our train set and not do so well in our testing validation accuracy. Furthermore, given that the third model had less parameters than the first but it took almost 1.5x more time to train for each epoch. This is due to it having a deeper architecture which would require a much longer time to train.

Suggestions

It is clear that these approaches might not be the best and have room for improvement. Some of suggestions would include trying out autoencoders for **dimensionality reduction** of the image or even trying **data augmentation**. We can also use some of the more complex and better performing model's parameters by applying **transfer learning** and further training a better model with better performance than the previous one.