# Deep Learning CNN Lab

Goktug Cengiz

Roberto Fernandez

October 2019

# Contents

**1 Introduction** **3**

**2 Data Preprocessing** **3**
2.1 Splitting the data into train, test and validation . . . . . 3
2.2 Normalization . . . . . 3
2.3 Reshaping . . . . . 3

**3 Modeling** **4**

**4 Scientific and personal conclusions** **8**

# 1 Introduction

The goal of the project is to classify the animals and to be able to identify their species. The data set[1] used in this project was provided by the Kaggle. It contains about 28K medium quality animal images belonging to 10 categories: dog, cat, horse, spider, butterfly, chicken, sheep, cow, squirrel and elephant. In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery. Therefore, we will use CNN in this study to achieve our goal.

# 2 Data Preprocessing

## 2.1 Splitting the data into train, test and validation

We divided the data set into 3 parts using the hold out method: training, validation and testing. % 70 of our total 28266 images was distributed as training, % 15 validation and % 15 testing. **train set** is the sample of data used to fit the model. Our model will see and learn from this data. **The validation set** is sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters. Our model occasionally sees this data, but never learn from this. It indirectly will affect to our model. **Test set** is sample of data used to provide an unbiased evaluation of a final model fit on the training data set. We will evaluate our model thanks to test set.

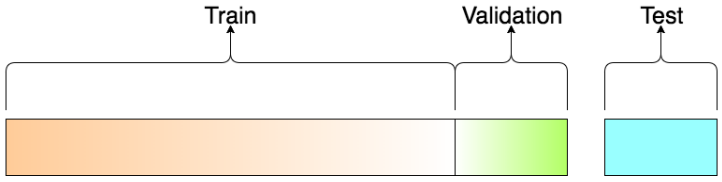


Figure 1: Data set splitting

## 2.2 Normalization

We performed a grayscale normalization to reduce the effect of illumination's differences and make CNN work faster.

## 2.3 Reshaping

Since our neural network will need a consistent shape of the images, we are forced reshape them before passing them to the network. We reshaped all data to 28x28x1 3D matrices.

# 3 Modeling

Once the preprocessing part is done, we used the VGG16 convolutional network to obtain a representation of the important features of the images. To increase the accuracy of our model, we need extra features apart from the colors of each pixel. These features can be very different based on what kind of dataset we are using and that is the reason we decided to use VGG16. This model is a convolutional neural network that achieves a very high accuracy in tests in ImageNet.

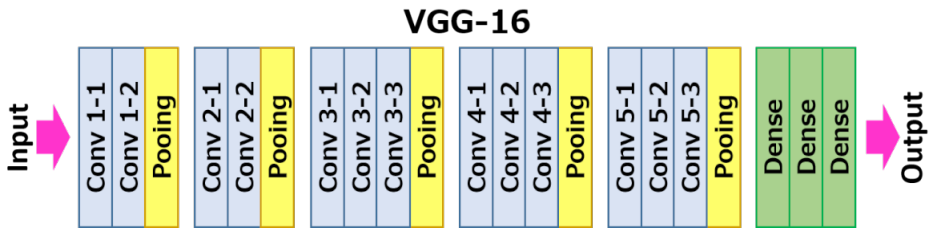


Figure 2: VGG16

VGG16's input layer is a 224x224 RGB image, which is passed through a series of filters to try to capture the most important features. The most important difference from the picture to what we actually used is that we did not use the dense layers in order to obtain the bottleneck features. In a visual way, it would be like removing the last 4 layers of the architecture and not using the fully connected layers and the softmax.

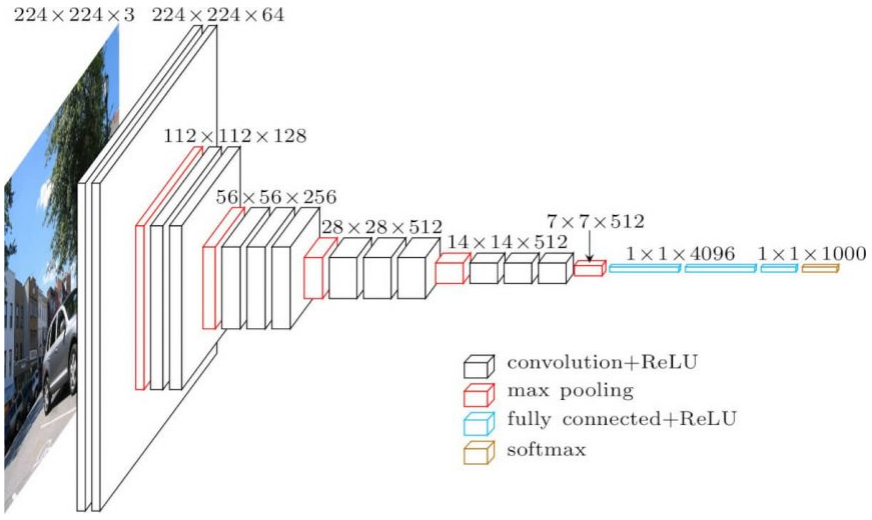


Figure 3: VGG16 Architecture

After using the VGG16 we will have the list of features we are going to use to train our model. We only need to define how to use them. We did a series of experiments to try to find the best combination:

---

Test 1

```
model = Sequential()
model.add(Flatten())
model.add(Dense(16, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer=optimizers.RMSprop(lr=1e-4), metrics=['acc'])
```

Performance:
Accuracy: 84.13% Loss: 0.5151807251737787 Time: 0:00:46.057089

---

Test 2

```
model = Sequential()
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer=optimizers.RMSprop(lr=1e-4), metrics=['acc'])
```

Performance:
Accuracy: 83.80% Loss: 0.5081063956834974 Time: 0:00:58.142787

---

Test 3

```
model = Sequential()
model.add(Flatten())
model.add(Dense(100, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer=optimizers.RMSprop(lr=1e-4), metrics=['acc'])
```

Performance:
**Accuracy: 85.57% Loss: 0.47512443192557974 Time: 0:02:31.182215**

---

Test 4

```
model = Sequential()
model.add(Flatten())
model.add(Dense(100, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(50, activation='relu'))
```

model.add(Dropout(0.3))
model.add(Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer=optimizers.RMSprop(lr=1e-4), metrics=['acc'])

Performance:
Accuracy: 83.49%  Loss: 0.52121547914444  Time: 0:02:23.758926

---

Test 5
model = Sequential()
model.add(Flatten())
model.add(Dense(100, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer=optimizers.RMSprop(lr=1e-4), metrics=['acc'])

Performance:
Accuracy: 85.54%  Loss: 0.4743479829597867  Time: 0:02:20.873357

---

Test 6
model = Sequential()
model.add(Flatten())
model.add(Dense(100, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(50, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer=optimizers.RMSprop(lr=1e-4), metrics=['acc'])

Performance:
Accuracy: 85.21%  Loss: 0.47360625791908156  Time: 0:02:22.548226

---

Test 7
model = Sequential()
model.add(Flatten())
model.add(Dense(100, activation=keras.layers.LeakyReLU(alpha=0.3)))
model.add(Dropout(0.3))
model.add(Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer=optimizers.RMSprop(lr=1e-4), metrics=['acc'])

Performance:

Accuracy: 85.42%  Loss: 0.5073217066391459  Time: 0:02:21.464939

---

Test 8

```
model = Sequential()
model.add(Flatten())
model.add(Dense(100, activation=keras.layers.LeakyReLU(alpha=0.3)))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer=optimizers.RMSprop(lr=1e-
4), metrics=['acc'])
```

Performance:

Accuracy: 84.93%  Loss: 0.4855860212536634  Time: 0:02:32.523399

---

Test 9

```
model = Sequential()
model.add(Flatten())
model.add(Dense(100, activation=keras.layers.LeakyReLU(alpha=0.5)))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer=optimizers.RMSprop(lr=1e-
4), metrics=['acc'])
```

Performance:

Accuracy: 84.58%  Loss: 0.4981114445429928  Time: 0:02:22.078212

---

Test 10

```
model = Sequential()
model.add(Flatten())
model.add(Dense(100, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

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

Performance:

Accuracy: 57.24%  Loss: 1.1583009414937135  Time: 0:01:34.437350

---

In the end, test 3 was the most successful with an accuracy rate of 85.57%, which is not very high compared to the other tests, but also had a low execution time with 58s.

# 4 Scientific and personal conclusions

We have presented a deep convolutional network for indoor scene recognition using animals-10 dataset. This dataset which was very suitable to deep learning, because it has no too many classes and it has many pictures per class. As we have seen above, we did 10 experiments. In some we've changed the dense numbers. We noticed that this did not change anything highly. We used two activations, rule and softmax. We achieved an accuracy of approximately 83-85. The most important difference is seen in the optimizer. When we lifted it, we could profit from speed, but the accuracy dropped to 57.

# References

[1] https://www.kaggle.com/alessiocorrado99/animals10

[2] https://www.towardsdatascience.com