Project1 - FNN and CNN

Emanuel Buchholz

March 22, 2019

Instructors: Dario Garca Gasulla

Important Links
Github
Bilinear Network
CAM

1 Introduction

For the first lab report about the practical applications of Convolutional Neural Networks, I choose the Standford Dog Dataset Khosla et al. (2011) which is a dataset containing 20,580 images of 120 breeds of dogs. This dataset has been built using images and annotation from the famous ImageNet dataset for the task of fine-grained image categorization. From this I build two convolutional neural networks, one rather simple one with only two convolutional layers and another one incorporating the pretrained 'VGG16' model that was trained on the original imagenet dataset. At the end I compared the results from these to experiments with each other.
Since these results were not satisfying, I looked into methods for fine-grained image classification and found the method of using bilinear CNNs for improving the accuracy of my network Lin et al. (2017).
In the end I also looked into which features my neural network picks up on. There are several strategies to visualize the convolutional layers. The most famous one is probably googles deep dream which allows for the visualization of every layer of GoogLeNet. I looked into the papers of Zeiler and Fergus (2013), Zhou et al. (2016) and Selvaraju et al. (2016). I decided on the second paper because it felt the easiest to implement. The only downside what that the method described in this paper required a special network structure and therefore I had to build another network which I trained on a malaria dataset that can be found here. It contains 27,558 images with two classes: infected/uninfected.

2 State of the Art/Methodology

Since CNNs have already been covered in the lecture, I will not describe them in detail. In its essence a CNN acts as a feature extractor that filters out irrelevant data. These features are then learned by the following an ANN. A very general CNN could look like in Figure 1. First the CNN part reduces the input data amount in matrix form and extracts important features. Then the features get flattened into a vector which can be feed to an ANN. The ANN then gives the classification or regression prediction.

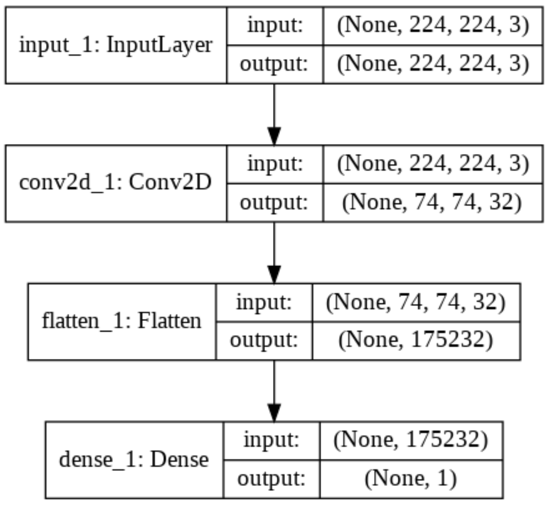


Figure 1: General structure of a CNN

For the implementation of my CNNs I choose a simple implementation, which I call the 'simple' network in the following, with two convolutional layers and two standard neural layers. This architecture can be seen in Figure 8.
Additionally I also build a network using the VGG16 model that is pretrained on the 'ImageNet' data as a convolutional layer and again two standard neural layers. I will call this model in the following VGG16. With this I hoped to achieve to compensate for the sparse amount of data in the Stanford Dog Dataset which only contains around 100-200 images per class. This method for compensating small datasets is quite common and many researchers use it as for example Lin et al. (2017). You can see the architecture of this model in Figure 9 in the appendix.

2.1 Fine-grained image recognition with bi-linear models

Achieving high accuracies with images of very similar classes is challenging, since the visual differences between classes can be overwhelmed by factors such as pose, viewpoint and location. Many solutions to these problems have been proposed, but mainly they are very complicated to implement. A method that seemed more easily to implement was proposed by Lin et al. (2017). Their proposed network consists of two convolutional feature extractors making the features bilinear and a linear classifier which in their case was a SVM but it can also be a neural network. As you can see in Figure 2, both convolutional networks process the input image. Then both their outputs get combined into a bilinear vector by multiplying them using the outer product. This vector can then be fed into a classifier such as a neural network. This type of network is good at modelling pairwise feature interactions while being transitionally invariant. This makes it ideal to pick up on the distinct features that differentiate for example bird species or in the case of my dataset dog breeds.

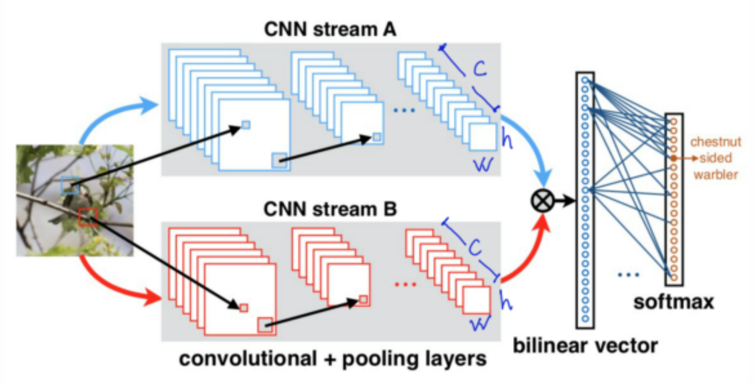


Figure 2: Proposed structure of a bilinear network as proposed by Lin et al. (2017)

You can see the implementation of my bilinear model in Figure 10 in the appendix. Here I again used the VGG16 model hoping the pretrained weights help with the small amount of images per class in my dataset. Then I tried to combine the output of the two layers using a custom function with a lambda layer for computing the outer product. Unfortunately this did not work and since I found several implementations using the dot function of keras, which corresponds to the inner product, I decided to give this a try. After this I connected the bilinear layer to a neural network with two layers as a classifier. The implementation of the bilinear network in keras was quite challenging and might have exceeded my knowledge of keras. Since I could not get the bilinear network to run on the cluster, I implemented it in Ggoogle Colab. The link can be found in the table of important links on the top as well as as notebook in the repository.

2.2 Network Visualization

CNNs are essentially black boxes that we feed our data to and they return a solution without us knowing how they found a solution or what features in an image they pick up on for classification of for example dog breeds. This can be done using Class Activation Maps (CAM)s. These are a simple technique to get the discriminative image regions used by a CNN to identify a specific class in the image. In other words, a class activation map (CAM) lets us see which regions in the image are relevant for recognizing a specific class. The authors of Zhou et al. (2016) show that this also allows re-using classifiers for getting good localization results, even when training without bounding box coordinates data. This shows that deep learning networks already have some kind of a built in attention mechanism.

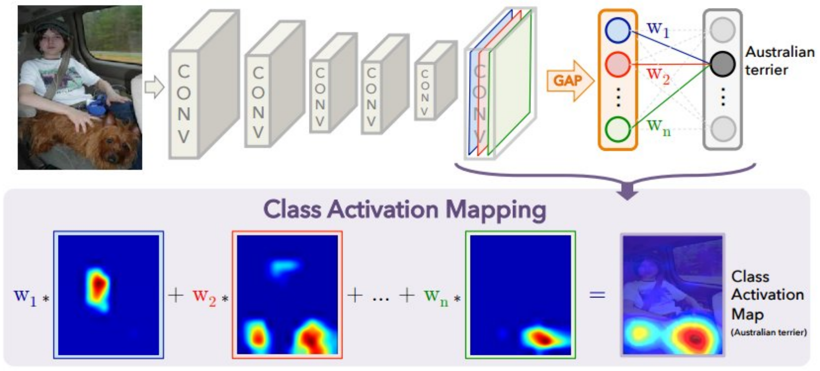


Figure 3: Class activation maps Zhou et al. (2016)

Unfortunately, in order to create a CAM, the network architecture is restricted to having a global average pooling layer after the final convolutional layer, and then a linear (dense) layer. Unfortunately this means it is impossible to apply this technique on existing networks that do not have this structure. Therefore I could not look at the activation maps of the networks I previously trained and had to create and train a new network. Since I wanted to keep this network as small as possible, and since I did not reach high accuracies I decided to use a different dataset with a large amount of data and only two classes. As said earlier I settled on the malaria dataset that can be found here and the network architecture in **??**. Additionally to the notebook from the repository the notebook can also be found in Google Colab with the link in the table at the top of the document.
It is also possible to use networks that do not follow this particular structure of a global average pooling layer + one dense layer. This is called Gradient-weighted Class Activation Mapping (Grad-CAM) and this method works on a wide variety of CNN families. It is described in Selvaraju et al. (2016), but I did not implement it.

3 Experimental Results

In the following section the experimental results will be explained. I have selected the most meaningful plots here, for further investigation have a look in the results folders of the repository for every network.

3.1 The Simple and VGG16 Models

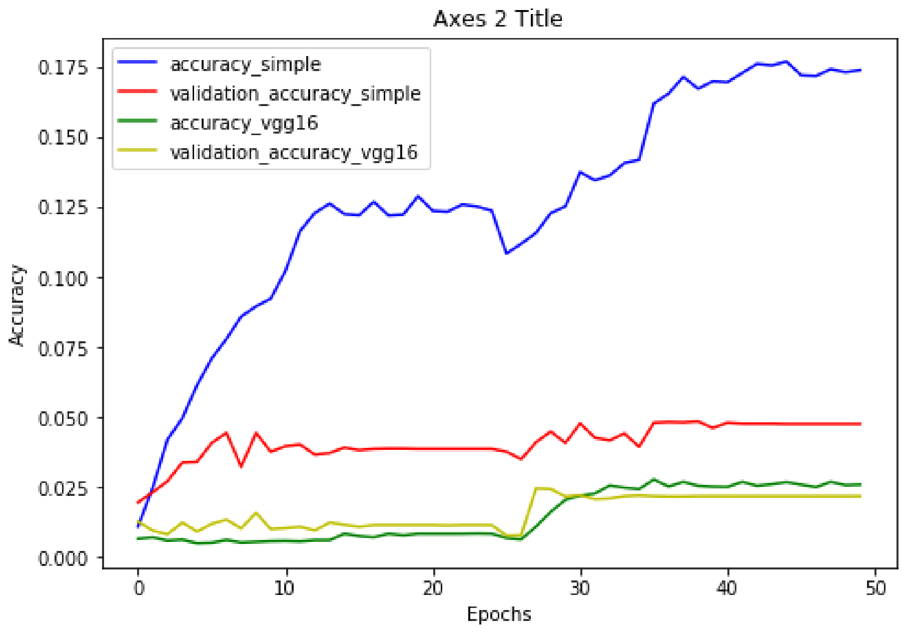


Figure 4: Accuracies and validation accuracies of the simple and VGG16 model

As you can see in Figure 4, both models did not reach a very high accuracy, with the simple model performing significantly better. The problem of the simple model is high overfitting. This problem might be solved by increasing the dropout value and including an additional dropout layer in the convolutional part. I suspect the additional dropout layer in the convolutional part would have been more important, since the VGG16 model does not show high overfitting and both networks have the same dropout layer in the neural network part.
My expectiation that including the pretrained VGG16 into my model would compensate for the lack of data did not come to fruition. But rather it made the model worse. I have no idea why this happened.
I trained the networks in two runs on 25 epochs each and as you can see reloading the network created some strange behaviour that I do not understand. It appears it kind of freed the model from a local minima, which makes me suspect that the saving and loading of the weights might have been erroneous and acted

similar to a network-wide dropout.
In general it showed that making a model more complicated and bigger does not make it better. But since more complicated models generally reach better values in the literature I think my lack of experience in building and training complex CNNs is to blame for the bad performance of the VGG16 model.

3.1.1 Confusion matrix

Since this classification task involves 120 classes, I omitted the classification report here. It can be viewed in the result folders of the respective networks. With the confusion matrix I created a heat-map for a more intuitive visualization. The heat-map of the simple model can be viewed in Figure 5. The VGG16 model is not displayed here, but can be viewed in the 'plot_etc.ipynb' notebook if one should be interested. But it just shows worse behaviour than the simple model.
An ideal confusion matrix would show a red diagonal line from the top left to the bottom right indicating correct predictions for all classes. Since this behaviour cannot be seen in the least, we know that our model is a really bad classifier. As you can see with the white streaks for some classes, essentially our model is biased towards some classes and ignores all the other classes. Looking at this heat-map I would say the model is far from being trained sufficiently and might have plateaued due to being stuck in a local minima. One way to address this could be to use a K-fold cross-validation to compensate for low amount of training values, fine-tuning of specific layers or a more complex model.

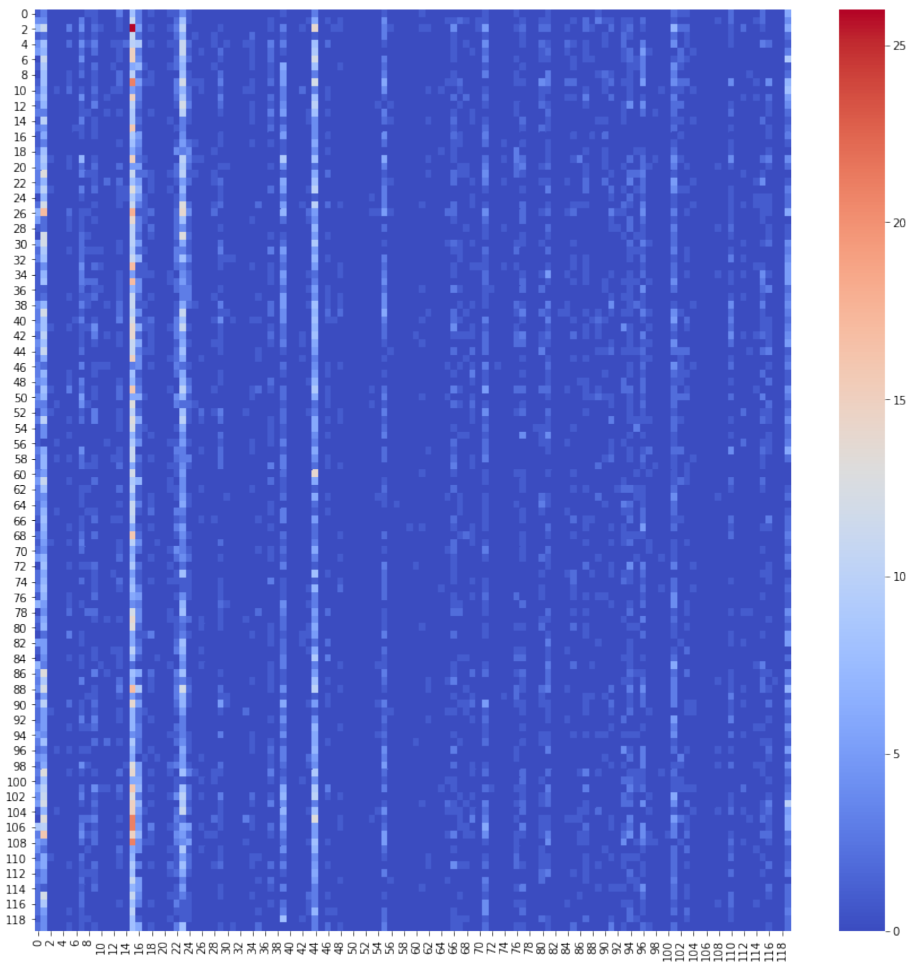


Figure 5: Confusion matrix of the simple model as heatmap

3.2 Bilinear Model

There is not much interpretation to be done for the bilinear model as you can see in Figure 6. Essentially the training for the network did not work and this is probably due to a wrong implementation of the network.

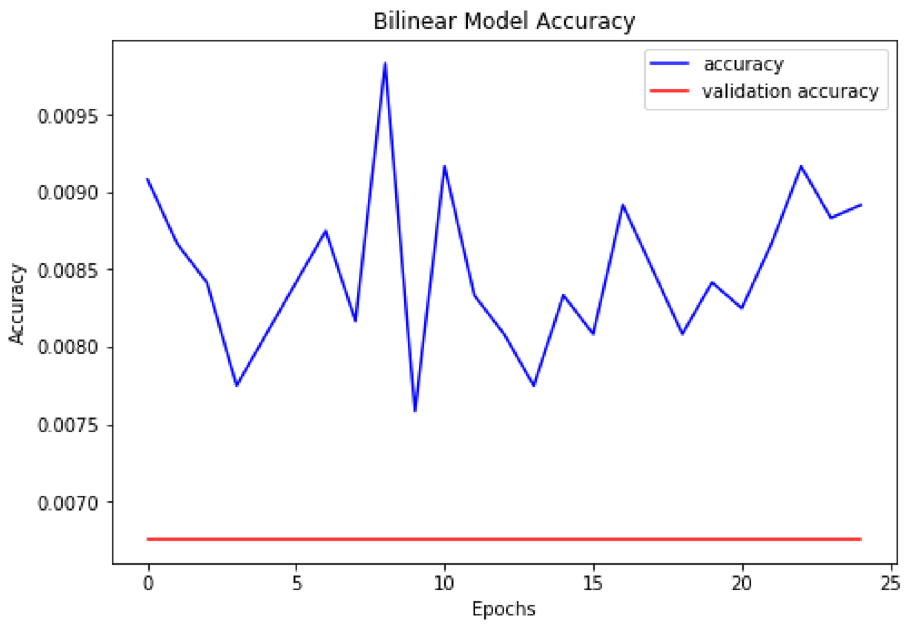
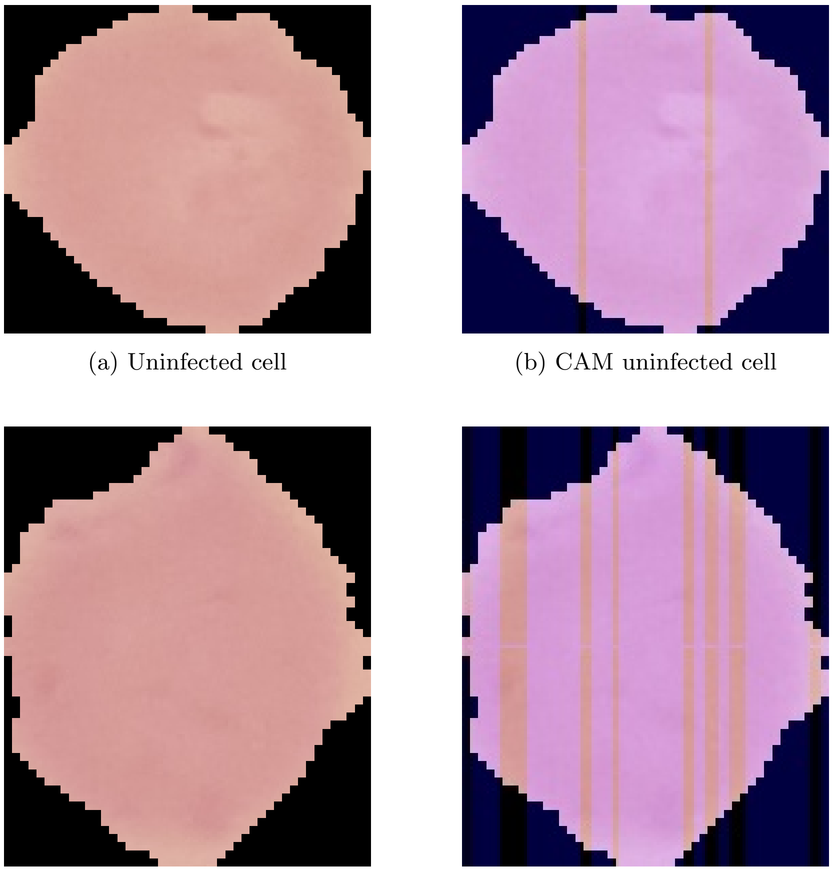


Figure 6: Accuracy of the bilinear model

3.3 Visualization with Discriminative Localization

In Figure 7 you can see two examples of a CAM for both classes of the malaria dataset. Since I do not know a lot about what features indicate malaria and pictures from both classes look quite similar to me, I am not sure what features the CNN pics up on. Since the network only reached an accuracy of 70% and due to the stripey pattern I suspect that the CAM is not extremely meaningful especially if compared to Figure 3.



(a) Uninfected cell (b) CAM uninfected cell

(c) Paracitized cell (d) CAM paracitized cell

Figure 7: Examples with CAM (right) of the two classes and without CAM (left)

4 Conclusions

In general I found that using more complex models does not necessarily improve the performance of a network and that it is usually the best to choose a simple model first before trying to improve its performance. Also the classification of similar image classes is a hard problem and a problem that is currently still under investigation. Therefore it might have been wiser to choose data-sets that bring clearer results. In general I found this lab work very interesting and challenging. Also I learned a lot about CNN network architectures and their capabilities, weaknesses and strengths if they are implemented correctly.

5 Appendix

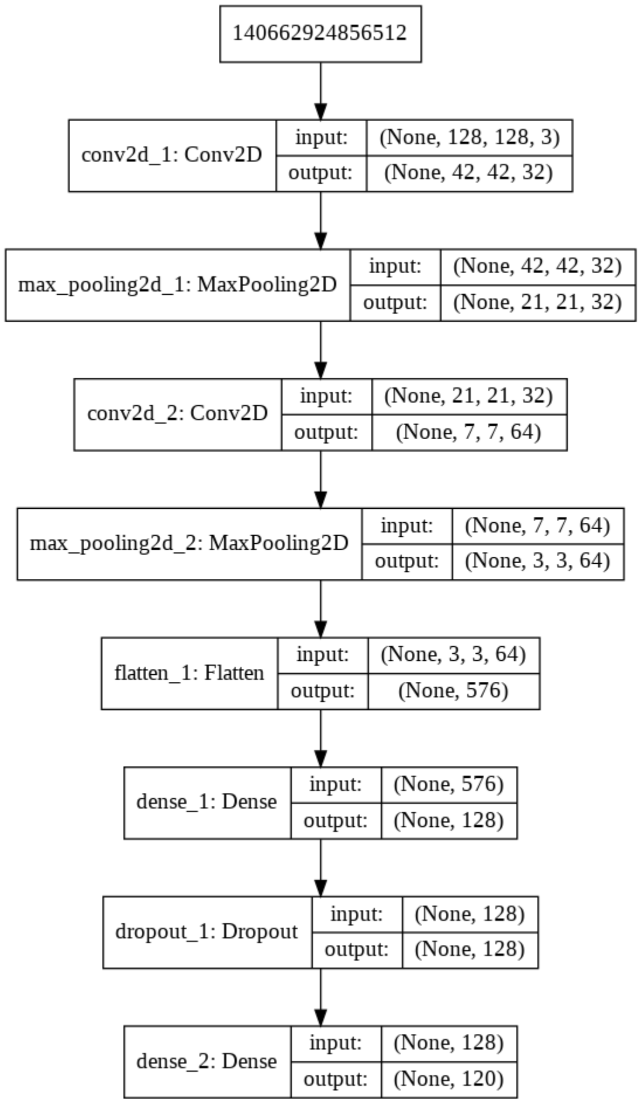


Figure 8: Structure of the simple model

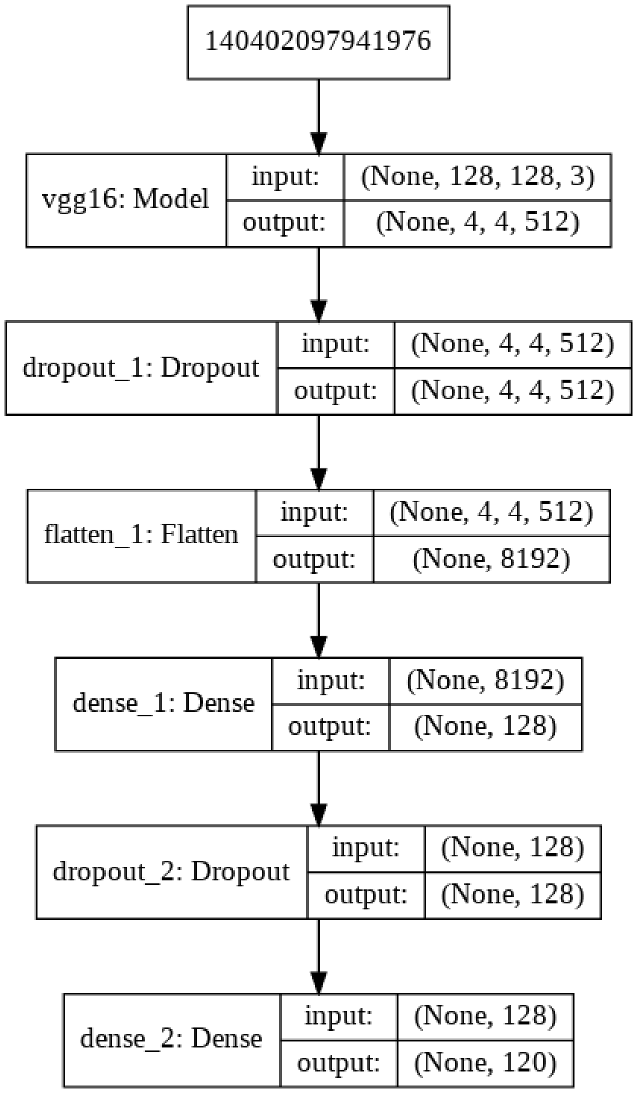


Figure 9: Structure of the pre-trained VGG16 model

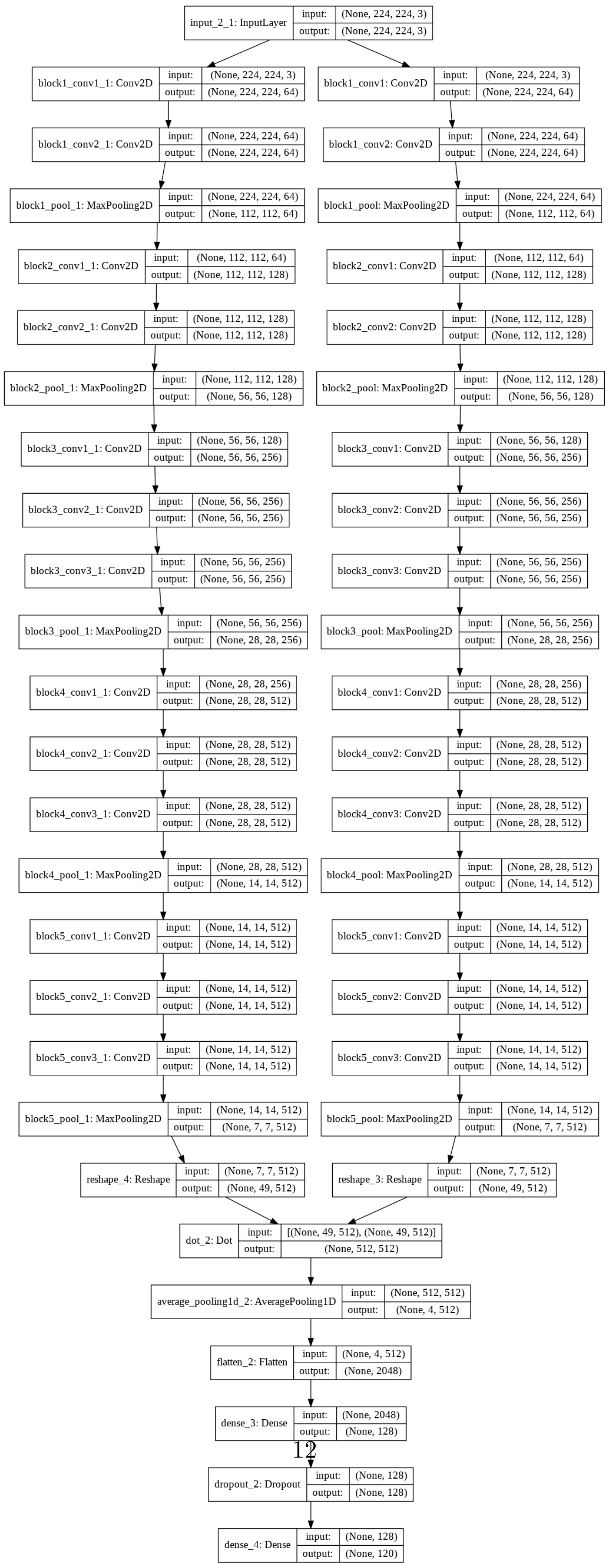

input_2_1: InputLayer input: (None, 224, 224, 3) output: (None, 224, 224, 3)
block1_conv1_1: Conv2D input: (None, 224, 224, 3) output: (None, 224, 224, 64)
block1_conv1: Conv2D input: (None, 224, 224, 3) output: (None, 224, 224, 64)
block1_conv2_1: Conv2D input: (None, 224, 224, 64) output: (None, 224, 224, 64)
block1_conv2: Conv2D input: (None, 224, 224, 64) output: (None, 224, 224, 64)
block1_pool_1: MaxPooling2D input: (None, 224, 224, 64) output: (None, 112, 112, 64)
block1_pool: MaxPooling2D input: (None, 224, 224, 64) output: (None, 112, 112, 64)
block2_conv1_1: Conv2D input: (None, 112, 112, 64) output: (None, 112, 112, 128)
block2_conv1: Conv2D input: (None, 112, 112, 64) output: (None, 112, 112, 128)
block2_conv2_1: Conv2D input: (None, 112, 112, 128) output: (None, 112, 112, 128)
block2_conv2: Conv2D input: (None, 112, 112, 128) output: (None, 112, 112, 128)
block2_pool_1: MaxPooling2D input: (None, 112, 112, 128) output: (None, 56, 56, 128)
block2_pool: MaxPooling2D input: (None, 112, 112, 128) output: (None, 56, 56, 128)
block3_conv1_1: Conv2D input: (None, 56, 56, 128) output: (None, 56, 56, 256)
block3_conv1: Conv2D input: (None, 56, 56, 128) output: (None, 56, 56, 256)
block3_conv2_1: Conv2D input: (None, 56, 56, 256) output: (None, 56, 56, 256)
block3_conv2: Conv2D input: (None, 56, 56, 256) output: (None, 56, 56, 256)
block3_conv3_1: Conv2D input: (None, 56, 56, 256) output: (None, 56, 56, 256)
block3_conv3: Conv2D input: (None, 56, 56, 256) output: (None, 56, 56, 256)
block3_pool_1: MaxPooling2D input: (None, 56, 56, 256) output: (None, 28, 28, 256)
block3_pool: MaxPooling2D input: (None, 56, 56, 256) output: (None, 28, 28, 256)
block4_conv1_1: Conv2D input: (None, 28, 28, 256) output: (None, 28, 28, 512)
block4_conv1: Conv2D input: (None, 28, 28, 256) output: (None, 28, 28, 512)
block4_conv2_1: Conv2D input: (None, 28, 28, 512) output: (None, 28, 28, 512)
block4_conv2: Conv2D input: (None, 28, 28, 512) output: (None, 28, 28, 512)
block4_conv3_1: Conv2D input: (None, 28, 28, 512) output: (None, 28, 28, 512)
block4_conv3: Conv2D input: (None, 28, 28, 512) output: (None, 28, 28, 512)
block4_pool_1: MaxPooling2D input: (None, 28, 28, 512) output: (None, 14, 14, 512)
block4_pool: MaxPooling2D input: (None, 28, 28, 512) output: (None, 14, 14, 512)
block5_conv1_1: Conv2D input: (None, 14, 14, 512) output: (None, 14, 14, 512)
block5_conv1: Conv2D input: (None, 14, 14, 512) output: (None, 14, 14, 512)
block5_conv2_1: Conv2D input: (None, 14, 14, 512) output: (None, 14, 14, 512)
block5_conv2: Conv2D input: (None, 14, 14, 512) output: (None, 14, 14, 512)
block5_conv3_1: Conv2D input: (None, 14, 14, 512) output: (None, 14, 14, 512)
block5_conv3: Conv2D input: (None, 14, 14, 512) output: (None, 14, 14, 512)
block5_pool_1: MaxPooling2D input: (None, 14, 14, 512) output: (None, 7, 7, 512)
block5_pool: MaxPooling2D input: (None, 14, 14, 512) output: (None, 7, 7, 512)
reshape_4: Reshape input: (None, 7, 7, 512) output: (None, 49, 512)
reshape_3: Reshape input: (None, 7, 7, 512) output: (None, 49, 512)
dot_2: Dot input: [(None, 49, 512), (None, 49, 512)] output: (None, 512, 512)
average_pooling1d_2: AveragePooling1D input: (None, 512, 512) output: (None, 4, 512)
flatten_2: Flatten input: (None, 4, 512) output: (None, 2048)
dense_3: Dense input: (None, 2048) output: (None, 128)
dropout_2: Dropout input: (None, 128) output: (None, 128)
dense_4: Dense input: (None, 128) output: (None, 120)

Figure 10: Structure of the bilinear model with two pre-trained VGG16 models

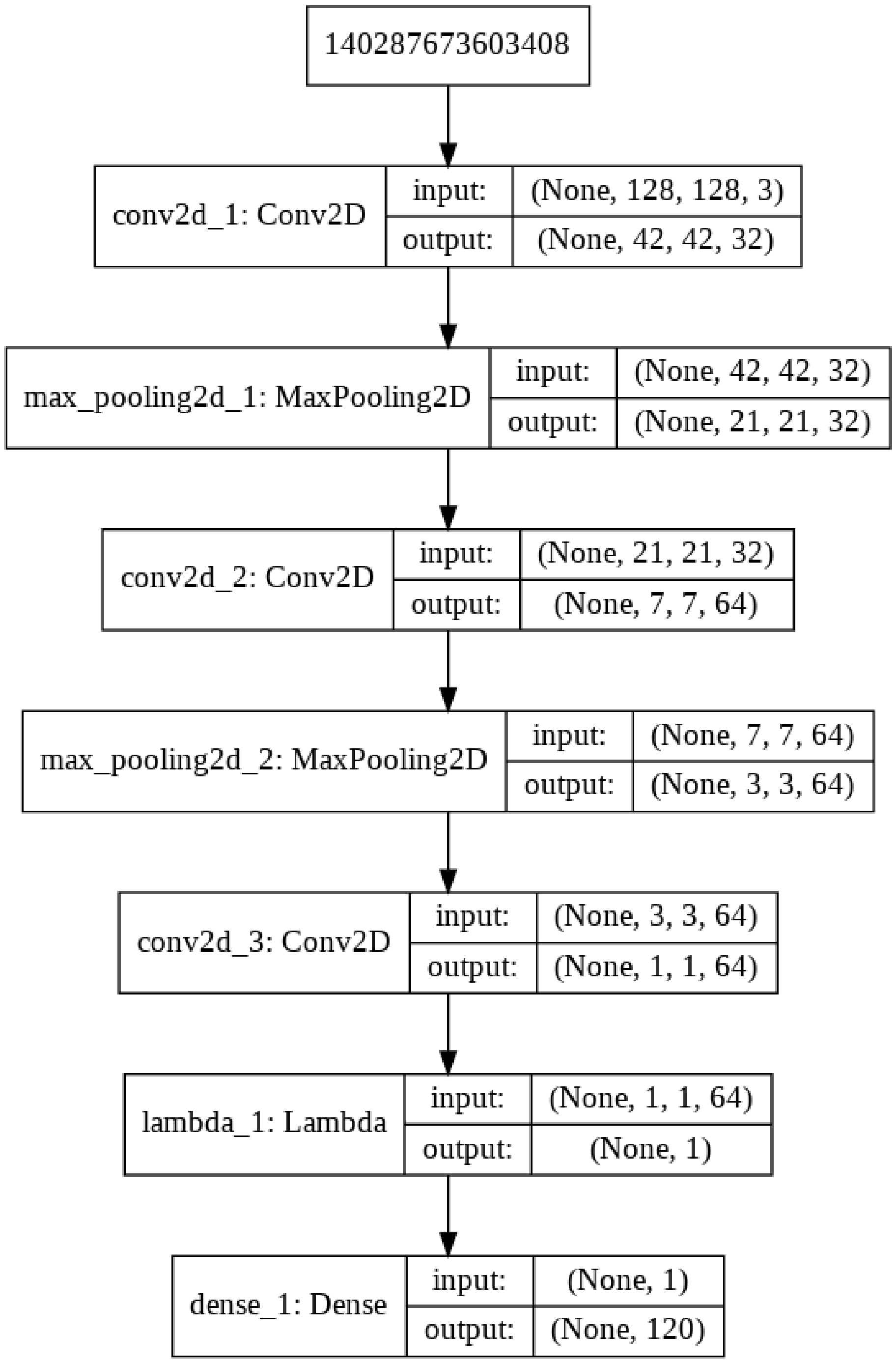


Figure 11: Network used for Class Activation Maps

References

Khosla, A., Jayadevaprakash, N., Yao, B., and Fei-Fei, L. (2011). Novel dataset for fine-grained image categorization. In *First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition*, Colorado Springs, CO.

Lin, T.-Y., RoyChowdhury, A., and Maji, S. (2017). Bilinear cnns for fine-grained visual recognition. In *Transactions of Pattern Analysis and Machine Intelligence (PAMI)*.

Selvaraju, R. R., Das, A., Vedantam, R., Cogswell, M., Parikh, D., and Batra, D. (2016). Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization. *CoRR*, abs/1610.02391.

Zeiler, M. D. and Fergus, R. (2013). Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901.

Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., and Torralba, A. (2016). Learning deep features for discriminative localization. In *Computer Vision and Pattern Recognition*.