

Context-Aware Pooling

<https://docs.google.com/document/d/1JfWm7A1o5fe245pdfr1SoF0uYcERg9hRThb5i4MyTmM/edit>

GitHub Repo: <https://github.com/AljonovMukhammaddiyor/CAP>

Introduction

General image classification—discrimination between people and boats, or cats and dogs—is a well-known and well-researched topic. However, a more interesting case is fine-grained image classification, where we classify between classes; for example, discriminating between breeds of dogs or species of birds.

Related Works

The paper “Context-aware Attentional Pooling (CAP) for Fine-grained Visual Classification” completed this task by attaching a “Context-aware Attentional Pooling” layer onto a pretrained deep CNN.

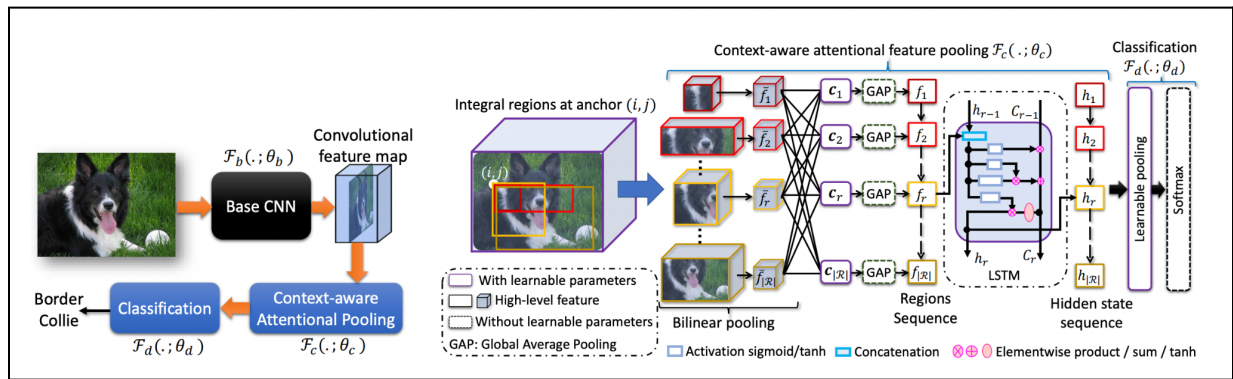


Illustration of the CAP model

The main idea is that the CAP layer considers contextual information at several scales. In theory (and as shown later on, in practice), it hierarchically captures information so that the subtle differences in features associated with fine-grained image classification can be modeled more effectively.

CAP Pipeline:

- Integral Regions
- Bilinear Pooling
- Context-Aware Attention
- Spatial Structure Encoding

Method

Our approach was to implement the context-aware attentional pooling (CAP) model proposed in the paper and replicate its results. The CAP module takes a convolutional feature from a base CNN and learns to emphasize the latent representation of multiple integral regions within the feature map. These regions are then fed into a recurrent network, such as an LSTM, to capture their spatial arrangements and produce a comprehensive feature vector for subordinate classification. We used the data generation and training functions from an available online implementation for efficiency, but implemented the CAP layer ourselves.

In more detail, the model uses an input image to predict a subordinate class label. It is trained on a set of many images, each with a fine-grained label, and aims to learn a mapping function that can accurately predict the label of a given image. This is done by minimizing a loss function between the true and predicted labels. The model has three components: a base CNN, a context-aware attentional pooling (CAP) module, and a classification module. The classification module uses a state-of-the-art CNN architecture.

We decided to attach the CAP layer to the Xception net.

After implementation, our plan was to first compare this CAP model to a couple baselines, mainly:

1. Xception net alone (Baseline 1)
 - a.

```
model = tf.keras.Sequential([  
    base,  
    tf.keras.layers.Flatten(),  
    tf.keras.layers.Dense(units=NUM_CLASSES, activation="sigmoid")  
])
```
2. Xception net + average pooling (Baseline 2)
 - a.

```
model = tf.keras.Sequential([  
    base,  
    tf.keras.layers.GlobalAveragePooling2D(),  
    tf.keras.layers.Flatten(),  
    tf.keras.layers.Dense(units=NUM_CLASSES, activation="sigmoid")  
])
```
3. Xception net + max pooling (Baseline 3)
 - a.

```
model = tf.keras.Sequential([  
    base,  
    tf.keras.layers.GlobalMaxPooling2D(),  
    tf.keras.layers.Flatten(),  
    tf.keras.layers.Dense(units=NUM_CLASSES, activation="sigmoid")  
])
```
4. Xception net + simple fully connected architecture (Baseline 4)
 - a.

```
model = tf.keras.Sequential([  
    base,  
    tf.keras.layers.Flatten(),  
    tf.keras.layers.BatchNormalization(),  
    tf.keras.layers.Dropout(0.1),  
    tf.keras.layers.Dense(units=128, activation="relu"),  
    tf.keras.layers.Dense(units=NUM_CLASSES, activation="sigmoid")  
])
```

We conducted these tests separately for efficiency; the CAP model implementation and training were done in Google Colab while the baselines were created and trained on Kaggle.

For the Kaggle implementation, the data loading functions were taken from another notebook, while the model creation and training were done by us.

In order to evaluate the effectiveness of our model, we planned to train it on multiple datasets and compare the results to those reported in the original paper. This would allow us to see how well our model performs in comparison and determine if it is a viable alternative. By training the model on multiple datasets, we can also evaluate its generalizability and ability to adapt to different data sets. Overall, this comparison will provide valuable insights into the performance and capabilities of our model, but due to time constraints, we were unable to implement and train these by now.

Challenges and Fixes

One of the challenges we encountered during training was overfitting. This occurred when the validation accuracy started to increase but the validation loss began to increase after a decrease. We suspect that this was due to the model becoming too complex and starting to memorize specific details of the training data rather than learning general patterns.

To address this issue, we tried implementing a learning rate scheduler to dynamically adjust the learning rate during training. This allowed the model to find a more optimal learning rate and potentially prevent overfitting.

The other issue was formatting the data so that it could be used with the data loader we found, particularly the file structure. We downloaded the dataset off of the Caltech website; all the images were split up into classes. However, the data loader required training and testing data to be split up into different directories. To deal with this, we wrote a script to reorganize the data (since this has nothing to do with the deep learning aspect of the project, the code for this will be omitted).

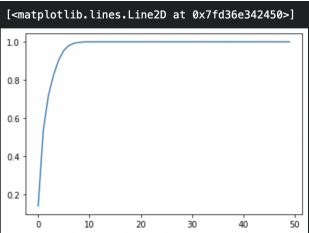
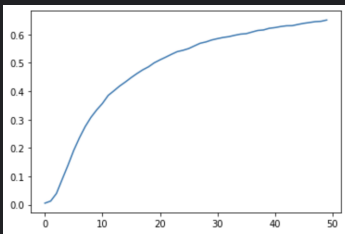
The next issue was that the data was too large to use for the CAP layer; the first epoch didn't finish even after an hour. As such, we decided to reduce the number of classes.

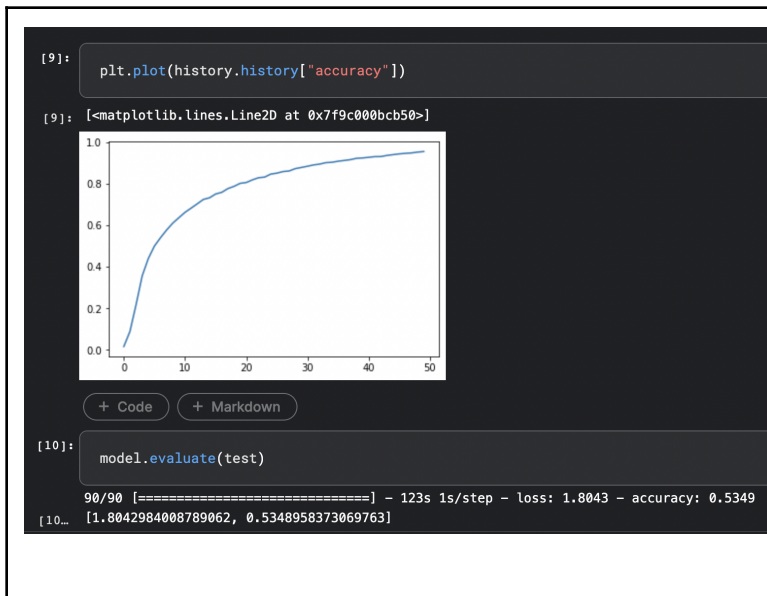
Results & Analysis

Accuracies for each model for CUB-200-2011 birds dataset

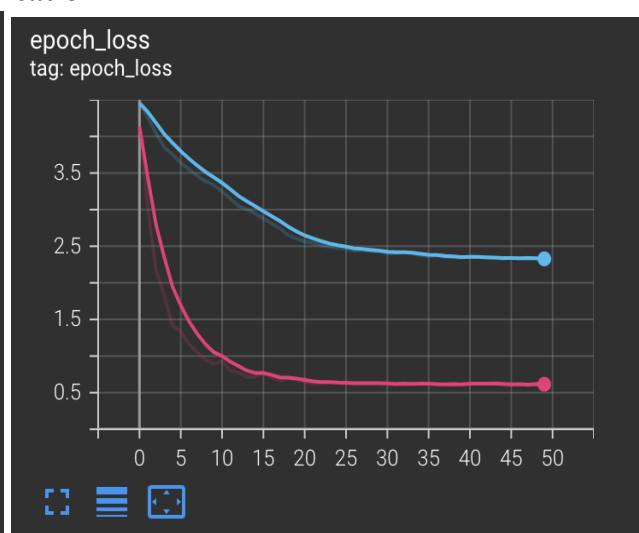
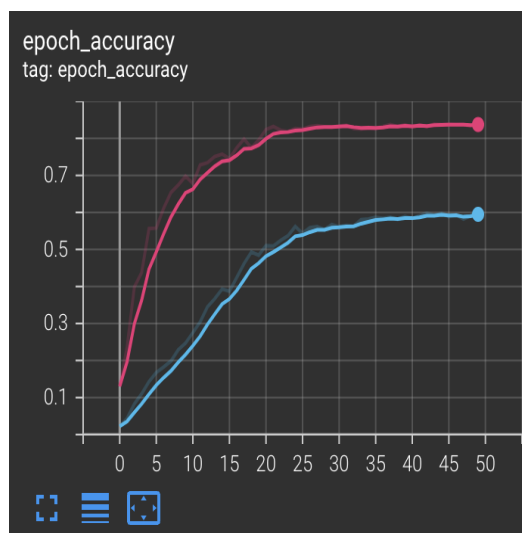
	Baseline 1	Baseline 2	Baseline 3	Baseline 4	CAP Replication	CAP Paper
Accuracy(%)	53.98	48.16	53.49	45.66	83.5	91.8

Accuracy per epoch for each model

Baseline 1	Baseline 2
<pre>[13_ <matplotlib.lines.Line2D at 0x7fd36e342450>] [14]: model.evaluate(test) 90/90 [=====] - 120s 1s/step - loss: 1.9604 - accuracy: 0.5398 [14_ [1.9604030847549438, 0.5397569537162781]</pre> 	<pre>[21]: plt.plot(history.history["accuracy"]) [21_ <matplotlib.lines.Line2D at 0x7fcfcc325c10>] [22]: model.evaluate(test) 90/90 [=====] - 114s 1s/step - loss: 2.3895 - accuracy: 0.4816 [22_ [2.38947916030838, 0.4815972149372101]</pre> 
Baseline 3	Baseline 4



CAP Replication



As can be seen from the accuracy table, though the four baseline models do an alright job for such a high-cardinality classification task, their performances pale in comparison to the CAP.

Overall, our model achieved a validation accuracy of 83.5% on 80 classes, which is relatively good compared to the 91.3% accuracy reported in the original paper on 200 classes. The use of a smaller number of images and a shorter training time likely contributed to the lower accuracy, but implementing the learning rate scheduler helped us improve the performance of the model. This helped to decrease the validation loss steadily during training. Previously, without the learning rate scheduler, the validation loss started to increase at a certain point, even though the training loss was still decreasing, as shown below.



One potential cause of this behavior could be overfitting. The model may be memorizing the training data, but not generalizing well to new, unseen data. The learning rate scheduler helps to prevent overfitting by adjusting the learning rate during training, which can lead to better generalization and improved performance on the validation set.

It is also possible that decreasing the number of classes from 200 to 80 compensated for the lower number of epochs or images to some degree. A smaller number of classes may have made it easier for the model to learn and generalize, leading to better performance on the validation set.

Compared to Xception, Xception with max pooling, and Xception with a fully connected architecture, the model with CAP performed much better. This indicates that the use of CAP to effectively encode the spatial arrangements and visual appearance of parts within an object really worked well. This allowed our model to better capture the hierarchical relationships within objects and their parts, leading to improved classification accuracy.

Conclusion

In conclusion, our experiments have shown that the context-aware attentional pooling (CAP) module can be effective at capturing the spatial arrangements and visual appearance of parts within an object, and using this information to improve fine-grained visual classification. By using a base CNN and the CAP module, we were able to achieve good performance on a reduced dataset with only 80 classes.

While our results are encouraging, there is still room for improvement. In future work, we plan to explore different base CNN architectures and training approaches to see if we can further improve the performance of our model. Additionally, we also plan to evaluate our model on the full dataset with 200 classes to see if it can generalize well to a larger number of classes. Overall, our experiments have shown that the CAP module is a promising approach for fine-grained visual classification.

Contributions

In this project, we were both responsible for implementing the base CNN and the context-aware attentional pooling (CAP) module. This included designing the architecture of the CAP module and implementing it in TensorFlow. Mukhammaddiyor performed training on the model with a CAP layer and obtained the results. Tivan performed experiments with four baselines.

We wrote the report together; Tivan wrote the introduction and related works, and Mukhammaddiyor wrote the conclusion and contributions. Other parts of the report, such as Method, Challenges and Fixes, and Results and Analysis, were contributed to by both of us.

Both of us contributed to the overall project by discussing the design and approach, providing suggestions and feedback, and collaborating on the implementation and experimentation. Overall, the collaboration between Tivan and Mukhammaddiyor allowed for a successful project that achieved good performance on a reduced dataset.

References

Referenced Paper:

-

Referenced Github Link:

- <https://github.com/ArdhenduBehera/cap>

Referenced Kaggle Link:

-