CS783 Assignment2

Harshit Sharma (160283)

Ashish Kumar (160160)

Introduction

Aim of this assignment was to do object categorization, given a dataset of images we need to categorize images into their coarse and fine classes. Each coarse classes is divide into some fine classes. In the following points we'll be describing about our approaches to achieve this task and the problems faced and subsequent improvements that led us to achieving the final model.

Approach for coarse grained classification

Approach

- We used the technique named **Transfer Learning** for training our dataset on coarse classes. Transfer learning is the use of pre-trained model on the new data, so here we used **VGG16 classfier** trained on **Image Net** dataset to classify our images on coarse classes.
- We used this method of learning because as deep networks contains millions of parameters and our dataset was quite small to train the parameters and it may lead to overfitting(case of very high training accuracy and low test accuracy). Thus Transfer Learning works great when we want to apply Deep Nets and training data available is very little.
- Now first we **randomly** divided our data as — 70% training , 15% validation , 15% test.
- We used **Keras** framework, first we loaded all the data into our model then did some **data preprocessing** such as **data augmentation**, which given it's arguments flips the training images by specified angle to generate new training images which can now be included in our training data enhancing it even further.
- Then we loaded the **pre-trained** VGG16 model with 138M parameters i.e too high, so we removed all it's fully connected layers (because maximum number of params are present in fully connected layers) which finally gave us the model with 14.7M params.
- Now over VGG model, we used 256 **conv filters** with 3x3 dimm, then applied **dropout** of 0.25, then flattened it so that we can apply 1 **dense fully connected layer** on it. So due to these improvisations our now our Trainable params become 1.2M.
- Training was performed on my laptop's GPU and we stopped the training once validation loss starts converging.
- For testing on data, we load the trained model and then predict coarse class using it.

Results

- Training Accuracy for coarse grained classification — 98%
- Validation Accuracy for coarse grained classification — 98%
- Test Accuracy for coarse grained classification — 99%

Adaptation and Improvements

- We used **data augmentation** technique to increase our training data so that we can get better classification.
- Used **Dropout** to decrease the number of paramters, as there was an upper bound to total number of params we can use.
- If good accuracy is achieved fast then we **reduced the number of epochs** to prevent overfitting.
- Made changes in the original VGG model and **added some new layers** according to the performance of the overall model and also to reduce the parameters of the overall model.

Approach for fine grained classification

Method

We'll need to make separate model for each class.Then given an image,first we'll be doing coarse grained classification for an image using the method used above, then according to the coarse class obtained we'll be doing fine grained classification using the model for that class. method is described as follows;-

- We divided the total data as —- 80% training , 10% validation , 10% test , as for each fine class we've quite limited data so we increased the proportion of training data in our model.
- For each coarse class we loaded the corresponding data and used it to train the model for fine classification which is as follows:-
  - First we used the pre-trained VGG16 model, then added 128 **conv filters**, after that added **Dropout** of 0.4 ,then flattened it and over it we applied **dense fully connected layer** with **128 nodes**, then applied **Batch Normalization** , then used **ReLu activation function** , again applied **Dropout** of 0.4 and then again applied **Final dense layer** with **number of nodes equal to number of classes** with **Softmax activation**.
  - **Categorical_crossentropy loss** is used for overall loss calculation and **Adam optimizer** is used.
- Training of our model is done on my laptop's GPU and training is stopped when validation loss starts to converge.
- For testing, first we predict the coarse class and then predict the fine class using fine grained classification model mentioned above.

Results of Test Accuracy

- Flowers –» 99% .
- Aircraft –» 95%.
- Dogs –» 96%
- Cars –» 81%
- Birds –» 74%

Adaptation and Improvements

- Initially we tried applying the coarse grained classification approach by just changing the number of classes to 36, but it only resulted in accuracy around 60% which was not satisfying.
- We used **data augmentation** technique which included rotation,shifting,flipping to generate new data. This is done to increase our training data size and it also resulted in increase of test accuracy.

- We also tried **vertical flipping** but it **doesn't increased our accuracy** so we rejected it. Note that we performed augmentation only in training data.
- We used **Dropout**=0.4 as it prevents over-fitting and also reduces the number of params as it drops out some of the connections between consecutive layers.
- We also **tried to train the last 2 layers** of our pre-trained VGG model, although it increased the accuracy but it also shoots up the number of parameters, thus we rejected this idea.
- We also added **Batch Normalization Layer** after **dense FC layer**, this resulted in increase of accuracy for all the fine classes.
- Still we were not able to achieve desired accuracy on the birds so we tried different methods e.g **tweaking the number of conv filters, number of nodes in dense FC layer, changing the number of epochs**, etc. Finally we applied **normalization to whole input data** before feeding it into the CNN, and this enhanced our **accuracy significantly from 55% to 74%**.
- Due to rise in accuracy by using this approach on birds data, we tried this on all of the fine classes and it enhanced our accuracy on all of the fine classes.

Number of Parameters

- **Common parameters in all models** 14.7M
- **Coarse_Parameters** 1.2M
- **Fine _Parameters**
  - **Flower_Parameters** 0.87M
  - **Aircraft_Parameters** 0.87M
  - **Cars_Parameters** 1.42M
  - **Dogs_Parameters** 1.35M
  - **Birds_Parameters** 1.75M
- **Total Parameters of our model** 22.16M
- **Maximum Number of parameters allowed** 11.7*2 = 23.4M

Hashes

- **coarse** : 3153daf7bb12a61dfed3b7484d4b1e52
- **flower** : 9962d78df61d00660b5168eb189fc786
- **aircrafts** : b5b68d886cdedc5eea6e8ac1c566ef86
- **dogs** : f8bd28efe18a7717236e8ca3230b12d0
- **cars** : 43da6649f1d7227c323abed6e73fe3d2
- **birds** : 8838ea04db35017a6666a7530ef9f130