BirdCaps : Bird species classification using Capsule Networks

Team members :

Yash Pradeep Gupte : A20472798 Namita Sandeep Tambe : A20460837

Abstract :

BirdCaps is a classification of the species of birds based on the novel architecture of Capsule networks. Convolutional Neural Networks (CNNs) are useful as they are able to accomplish complex tasks like image processing and computer vision. But there are some drawbacks of CNN's which include loss of spatial orientation. CNN disregards the placement of the image with respect to the image boundary and other relevant features. CapsNet overcomes this drawback of CNN. The capsule network in the BirdsCaps will capture the important spatial features of birds to differentiate amongst the different bird class labels. Different parameters of the images such as position, size, orientation, texture etc are compared to each other and stored in different levels of the capsule.We propose to construct a Capsule network model and evaluate its performance and compare and contrast with CNN model[1].

1. Introduction

1.1 Problem Statement :

To classify Bird species based on a novel architecture of Capsule networks. CapsNet, have shown major improvements in the field of computer vision when compared with conventional CNNs.

1.2 Proposed solution :

We propose to utilize Capsule Networks technique on a subset of the Caltech USD Birds dataset , and compare it with Convolutional Neural Networks. We perform data preprocessing and data augmentation on original dataset to reduce the number of classes and increase the per class image count. We have implemented a Dynamic Routing Algorithm of capsule networks and a novel Squash activation function to perform an image classification task. We have designed a Convolutional Network to show how it performs against capsule networks. The motivation of this project is to present a new way to look at image classification and how Capsule Network over performs CNNs.

2. Review of literature

Convolutional Neural Networks(CNNs) have shown brilliant computations when it comes to image processing and computer vision. CNNs have the ability to learn features with an element of spatial invariance. Object recognition accuracy of CNNs is the best , but there are some notable drawbacks as well. The MaxPooling operation used in CNN disregards the spatial orientation of involved features. The criteria to classify an image is the mere presence of a necessary object while disregarding its placement with respect to image boundary and other present features.

To solve this critical flaw of CNNs, Geoffrey Hinton proposed a solution called “routing-by-agreement” wherein lower level features will only get sent to a higher level layer that can match its contents. So if a lower level layer’s features resemble leaves and a bark and roots, it will be forwarded to a higher level layer that has identified a “tree” and so on.
Thus, Hinton proposed a complete solution that encoded spatial information into features and also used dynamic routing.

In order to implement this solution, the network architecture is adapted into a structure that groups together neurons with activity vectors representing similar spatial information, with the length of the activity vector indicating the probability of the feature existing in the image. The direction of the vector would represent it’s pose information. Each of these newly formed groups of neurons are called Capsules.

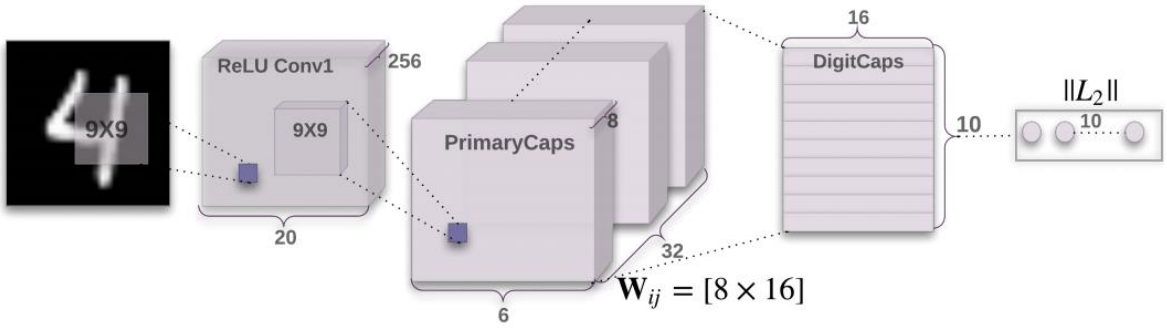
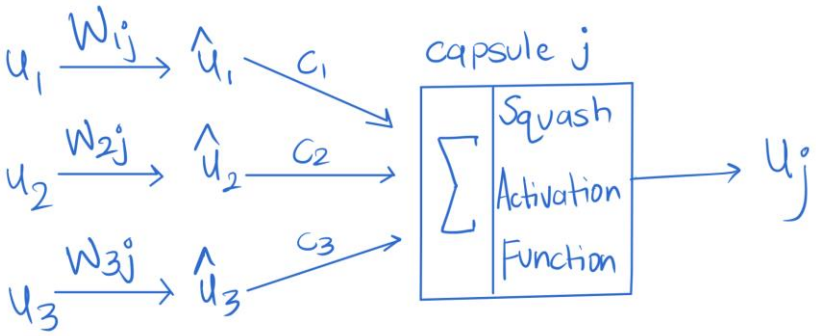


Figure 1[1]

Dynamic Routing Algorithm
Dynamic routing algorithm is the basis of capsule networks. That is how lower level capsules pass information to higher level capsules.



From the above image, when our current Capsule j receives Input vectors u1 , u2 and u3 where length of these vectors denote the probability of that corresponding object and direction of these vectors stores the internal state of these objects. For instance, if a low level capsule detects eye, mouth and nose and if the higher level Capsule j predicts Face.
These vectors are then multiplied by the weights (W1j , W2j , W3j) . For example matrix W1j encodes the relationship between eye and face and same for other low level capsules. After multiplication we get the predicted position of the higher level feature. In other words, $\hat{u}1$ represents where the face should be according to the detected position of the eyes, $\hat{u}2$ represents where the face should be according to the detected position of the mouth and $\hat{u}3$ represents where the face should be according to the detected position of the nose. If these predictions point towards the same position and state of face
Scalar weighing (c1, c2, c3)in the case of capsules , are decided using a dynamic routing algorithm.

Squash Activation function : This is a novel activation function which takes the output of a vector and squashes it to have length not more than 1, without changing the vector's direction.

$$\mathbf{v}_j = \frac{||\mathbf{s}_j||^2}{1 + ||\mathbf{s}_j||^2} \frac{\mathbf{s}_j}{||\mathbf{s}_j||}$$

Figure 2 : Squash Function [1]

where *vj* is the vector output of capsule *j* and *sj* is its total input

The right side of the equation scales the input vector so that it will have unit length and the left side performs additional scaling . Remember that the output vector length can be interpreted as the probability of a given feature being detected by the capsule.

Procedure 1 Routing algorithm.

1: **procedure** ROUTING($\hat{\boldsymbol{u}}_{j|i}, r, l$)
2: for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow 0$.
3: **for** r iterations **do**
4: for all capsule i in layer l: $\mathbf{c}_i \leftarrow \texttt{softmax}(\mathbf{b}_i)$ ▷ `softmax` computes Eq. 3
5: for all capsule j in layer $(l+1)$: $\mathbf{s}_j \leftarrow \sum_i c_{ij}\hat{\mathbf{u}}_{j|i}$
6: for all capsule j in layer $(l+1)$: $\mathbf{v}_j \leftarrow \texttt{squash}(\mathbf{s}_j)$ ▷ `squash` computes Eq. 1
7: for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow b_{ij} + \hat{\mathbf{u}}_{j|i}.\mathbf{v}_j$
return $\mathbf{v}_j$

Figure 3 : Dyanmic Routing Algorithm [1]

Explanation of the algorithm:

Line 1 : takes all the capsules in lower layer *l* and their output $\hat{u}$, as well as the number of routing iterations *r*.

Line 2 : a coefficient *bij* with a temporary value is declared which will be updated iteratively and after procedure is over its value is stored in *cij*. Initially at the start of training the value of *bij* is initialized to zero

Line 3 : steps in line 4 to 7 will be repeated *r* times . (number of routing algorithms)

Line 4 : calculates the value of vector *ci* which is all routing weights for a lower level capsule *i*. Softmax is applied so that *cij* is a non negative number and their sum equals one.

Line 5 : Here we look at higher level capsules. This step calculates a linear combination of input vectors , weighted by routing coefficients *cij* , this basically means scaling down input vectors and adding them together , which produces output vector *sj*.

Line 6 : vectors from the last step are passed through the squash nonlinearity and output vector *vj* for all the higher level capsules.

Line 7 : algorithm produces the output vector of higher level capsule *vj*.

3 Analysis and Design

3.1 Dataset

We are using the *Caltech - UCSD Birds 200 - 2011* dataset which is published by California Institute of technology, Caltech which consists of 200 categories of birds and 11,788 images [2][3].

3.2 Data preprocessing

The bird.zip data is extracted which contains test and train folders which individually contain the filenames, class_ids and embeddings. We then merge the train and test dataset respectively in the form of lists.

Next we merge the filenames.txt and bounding_box.txt files from the original dataset into a list. We then manually select the images which we want to save into a file and then we make a list of the selected images which consists of selected class_ids, filenames and embeddings.

Sort the selected images list and then save this list in a file.

Finally, we have 3976 train examples and 1196 test examples which gives a total of 5172 examples in the dataset.
Next we reduce the 200 categories of birds and club them under a superset. Following are the final 15 categories with the subsets of the birds that are considered for training and testing [2][3].

Table 1 : Reduced Categories - CUB 15[2][3].

Label Number	**Main Label Name**	**Subclasses from Original Dataset**
1	Blackbird_235_4	009.Brewer_Blackbird 010.Red_winged_Blackbird 011.Rusty_Blackbird 012.Yellow_headed_Blackbird)
2	Finch_239_4	034.Gray_crowned_Rosy_Finch 035.Purple_Finch 047.American_Goldfinch 048.European_Goldfinch
3	Flycatcher_417_7	037.Acadian_Flycatcher 038.Great_Crested_Flycatcher 039.Least_Flycatcher 040.Olive_sided_Flycatcher 041.Scissor_tailed_Flycatcher 042.Vermilion_Flycatcher 043.Yellow_bellied_Flycatcher
4	Grebe_240_4	050.Eared_Grebe 051.Horned_Grebe 052.Pied_billed_Grebe 053.Western_Grebe
5	Grosbeak_240_4	054.Blue_Grosbeak 055.Evening_Grosbeak 056.Pine_Grosbeak 057.Rose_breasted_Grosbeak
6	Gull_469_8	059.California_Gull 060.Glaucous_winged_Gull 061.Heermann_Gull 062.Herring_Gull 063.Ivory_Gull 064.Ring_billed_Gull 065.Slaty_backed_Gull 066.Western_Gull

7	Kingfisher_300_5	079.Belted_Kingfisher 080.Green_Kingfisher 081.Pied_Kingfisher 082.Ringed_Kingfisher 083.White_breasted_Kingfisher
8	Oriole_239_4	095.Baltimore_Oriole 096.Hooded_Oriole 097.Orchard_Oriole 098.Scott_Oriole
9	Sparrow_480_8	114.Black_throated_Sparrow 118.House_Sparrow 121.Grasshopper_Sparrow 127.Savannah_Sparrow 128.Seaside_Sparrow 129.Song_Sparrow 130.Tree_Sparrow 132.White_crowned_Sparrow
10	Swallow_239_4	135.Bank_Swallow 136.Barn_Swallow 137.Cliff_Swallow 138.Tree_Swallow
11	Tern_418_7	141.Artic_Tern 142.Black_Tern 143.Caspian_Tern 144.Common_Tern 145.Elegant_Tern 146.Forsters_Tern 147.Least_Tern
12	Vireo_409_7	151.Black_capped_Vireo 152.Blue_headed_Vireo 153.Philadelphia_Vireo 154.Red_eyed_Vireo 155.Warbling_Vireo 156.White_eyed_Vireo 157.Yellow_throated_Vireo
13	Warbler_480_8	158.Bay_breasted_Warbler 159.Black_and_white_Warbler 162.Canada_Warbler 164.Cerulean_Warbler

		165.Chestnut_sided_Warbler 167.Hooded_Warbler 176.Prairie_Warbler 182.Yellow_Warbler
14	Woodpecker_348_6	187.American_Three_toed_Woodpecker 188.Pileated_Woodpecker 189.Red_bellied_Woodpecker 190.Red_cockaded_Woodpecker 191.Red_headed_Woodpecker 192.Downy_Woodpecker
15	Wren_419_7	193.Bewick_Wren 194.Cactus_Wren 195.Carolina_Wren 196.House_Wren 197.Marsh_Wren 198.Rock_Wren 199.Winter_Wren

We then change the labels number of these categories from 1-15 to 0-14 to perform one hot encoding.

3.3 Data Augmentation

We have a total of 5172 images which results in overfitting of the training data and the accuracy of the testing data under fits the model. So we perform data augmentation to prevent overfitting of the training data.
Here we are rotating images, shifting the width, shifting the height, flipping the image and zooming the image.
This augmentation gives us 51720 extra images which we add with original images which gives a total of 56892 images where the final training and testing is performed [2][3].

3.2 System Architecture / Design

3.2.1 Capsule Network Model Architecture

```
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 64, 64, 3)]  0
__________________________________________________________________________________________________
Conv_layer_1 (Conv2D)           (None, 56, 56, 256)  62464       input_1[0][0]
__________________________________________________________________________________________________
leaky_re_lu (LeakyReLU)         (None, 56, 56, 256)  0           Conv_layer_1[0][0]
__________________________________________________________________________________________________
batch_normalization (BatchNorma (None, 56, 56, 256)  1024        leaky_re_lu[0][0]
__________________________________________________________________________________________________
Primarycaps_conv_layer_2 (Conv2 (None, 24, 24, 256)  5308672     batch_normalization[0][0]
__________________________________________________________________________________________________
reshape_primarycap (Reshape)    (None, 18432, 8)     0           Primarycaps_conv_layer_2[0][0]
__________________________________________________________________________________________________
squash_primarycaps (Lambda)     (None, 18432, 8)     0           reshape_primarycap[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 18432, 8)     32          squash_primarycaps[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 147456)       0           batch_normalization_1[0][0]
__________________________________________________________________________________________________
u_hat (Dense)                   (None, 240)          35389680    flatten[0][0]
__________________________________________________________________________________________________
softmax1 (Activation)           (None, 240)          0           u_hat[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 240)          57840       softmax1[0][0]
__________________________________________________________________________________________________
multiply (Multiply)             (None, 240)          0           u_hat[0][0]
                                                                 dense[0][0]
__________________________________________________________________________________________________
leaky_re_lu_1 (LeakyReLU)       (None, 240)          0           multiply[0][0]
__________________________________________________________________________________________________
softmax2 (Activation)           (None, 240)          0           leaky_re_lu_1[0][0]
__________________________________________________________________________________________________
multiply_1 (Multiply)           (None, 240)          0           u_hat[0][0]
                                                                 softmax2[0][0]
__________________________________________________________________________________________________
leaky_re_lu_2 (LeakyReLU)       (None, 240)          0           multiply_1[0][0]
__________________________________________________________________________________________________
softmax3 (Activation)           (None, 240)          0           leaky_re_lu_2[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 240)          57840       softmax3[0][0]
__________________________________________________________________________________________________
multiply_2 (Multiply)           (None, 240)          0           u_hat[0][0]
                                                                 dense_1[0][0]
__________________________________________________________________________________________________
leaky_re_lu_3 (LeakyReLU)       (None, 240)          0           multiply_2[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 15)           3615        leaky_re_lu_3[0][0]
==================================================================================================
Total params: 40,881,167
Trainable params: 40,880,639
Non-trainable params: 528
__________________________________________________________________________________________________
None
```

3.2.2 CNN Model Architecture

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 64, 64, 3)]       0
_________________________________________________________________
conv2d (Conv2D)              (None, 64, 64, 64)        1792
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 32, 32, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 32, 32, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 32)        18464
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 32)        9248
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 32)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 8, 8, 32)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 8, 8, 16)          4624
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 16)          0
_________________________________________________________________
dropout_3 (Dropout)          (None, 4, 4, 16)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 256)               0
_________________________________________________________________
dense_4 (Dense)              (None, 512)               131584
_________________________________________________________________
dense_5 (Dense)              (None, 15)                7695
=================================================================
Total params: 173,407
Trainable params: 173,407
Non-trainable params: 0
_________________________________________________________________
None
```

4. Implementation :

4.1 Implementation Details

4.1.1 Capsule Network Model

First Convolutional Layer

The 'build_capsule_module' is the function which we use to build the capsule model. We provide a 64*64*3 image size as input. A simple convolutional layer with 256 filters, 9 pixel kernel size and a stride of 1 comes as the first layer. The output of this layer is passed to the Primary Capsule layer. Higher number of filter sizes means that it increases the information capture ability of the subsequent layers. LeakyReLU and Batch Normalization is then applied which takes care of vanishing gradients [1].

Capsule Network

We define a Primary Capsule layer which is made up of 8D vectors of 32 feature maps. A reshape function is applied as it will split all output neurons into the 8D vector. At this stage the primary capsules have collections of activations which denote the orientation of the digit while intensity of the vector which denotes the presence of the each class. Finally a squash function and Batch normalization is applied. Squash function is to ensure that the length of the vectors is between 0 and 1. Batch Normalizing normalized the input to each unit to have zero mean and unit variance.

Next we have the 'Digit Caps' layer which implements the 'routing by agreement' algorithm. Here ,The output of Digit Caps is flattened neurons instead of passing vectorized values. This flattened neuron acts as a prediction vector. For 15 classes each with 16d vector we have 15*16 = 240 neurons.The prediction vector is then passed to the 3 length loop where a coupling coefficient which is a softmax over the bias weights of a previous layer and the prediction vector which is followed by LeakyRelu activation.

The last layer determines the prediction score of an image class , a dense layer with a 15 neuron and a softmax activation function [1] [4].

4.1.2 CNN model

The 'build_cnn_model' is the function which is used to build the simple CNN model. We provide a 64*64*3 image size as input. Then the model is followed with 4 blocks ofConventional Conv2d, max pooling layer and dropout layer with different hyper parameters. AT the end, we flatten the output and the output is passed to a dense layer with softmax activation and 15 output units each for 1 output class.

4.2 Coding Standard - keras python

Platform used for implementation: We have used Google Colab as a platform for training and implementing our Models. Google Colab offers a free runtime of 12 hours. It supports GPU as well as TPU(tensor processing unit) computational modules. Thus the tedious and time-consuming task of running complex models on local machines(CPU) is eliminated by adapting to such services.

Keras: Keras is a powerful easy-to-use Python library for developing and evaluating deep learning models. It warps the efficient numerical computation libraries CNTK and TensorFlow and allows you to define and train neural network models in a few short lines of code. It was developed with a focus on

enabling fast Being able to go from idea to result with the least possible delay is key to doing good research.

5. Result and Discussion

5.1. Discussion

Initially when we trained our model on 5172 images, we observed that the model was overfitting. Due to this we got test accuracy of 20% even when our train accuracy was 98%. Thus we performed data augmentation and combined the augmented images (10 transformations of each image total 51720 images) with the original dataset (5172 images) and got 56892 images [2].
We trained the model on the final augmented dataset for 60 epochs we got train accuracy and 97.83% and test accuracy of 91.48%.
We also trained the same set of images on a CNN model for 500 epochs. Although this CNN can be improved by adding more layers and fine tuning the hyper parameters.

5.2 Results

Following are the 6 images whose labels are predicted using the capsule network.

Image number	Actual Label	Predicted Label for Capsule network	Predicted Label for simple CNN model
Image 1 label	6	6	4
Image 2 label	8	8	15
Image 3 label	15	12	4
Image 4 label	8	8	11

Image 5 label	12	12	14
Image 6 label	4	4	7

The training accuracy for the capsule network recorded here is 97.83% whereas the test accuracy recorded is 91.48%
The training accuracy for simple CNN network recorded here is 68.59% whereas the test accuracy recorded is 60.30% [4].

6. Limitations

The sole reason for not utilizing the entire CUB 2002 2011 is that Capsule Networks are computationally expensive as it captures spatial features. The per class image count matters the most for capsule networks. We could have trained capsule networks with all the 200 classes but that would have required more GPU resources which were not available on Google Colab Platform. Hence, we decided to drill down the number of classes and performed data augmentation to increase the number of images per class.
Also, while training capsule networks we were not able to run in a single go, thus we had to save and load model after every 10 epochs. Hence we were not able to generate train validation and test accuracy & loss graphs during runtime [4].

7. Conclusion and Future work

7.1 Conclusion

The capsule network is more accurate compared to the CNN network. Even with a small number of hyperparameters, Capsule networks can efficiently classify the bird species. Capsule network requires more samples per class to achieve lower test error. It is not always necessary that Capsule networks will always give high accuracy compared to CNN networks , there are still better CNN networks than capsule networks as this research topic is still developing. The main motive of our project was to highlight the flaws in CNN and present how novel architecture of capsule networks perform on a different dataset.

7.2 Future work

The future scope of this project will be to try different datasets where the number images per class is higher. There exist multiple solutions to produce the computational workload of capsule networks. This will make it easy to execute this novel architecture on a huge amount of dataset

8. References

[1] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic Routing Between Capsules," *Adv. Neural Inf. Process. Syst.*, vol. 2017-December, pp. 3857–3867, Oct. 2017, Accessed: May 04, 2021. [Online]. Available: http://arxiv.org/abs/1710.09829.

[2] "Caltech-UCSD Birds-200-2011." http://www.vision.caltech.edu/visipedia/CUB-200-2011.html (accessed May 04, 2021).

[3] "birds.zip - Google Drive." https://drive.google.com/file/d/0B3y_msrWZaXLT1BZdVdycDY5TEE/view (accessed May 04, 2021).

[4] R. Mukhometzianov and J. Carrillo, "CapsNet comparative performance evaluation for image classification," May 2018, Accessed: May 04, 2021. [Online]. Available: http://arxiv.org/abs/1805.11195.