Comparison of Two Supervised Learning Methods for the Classification of Jellyfish Images

Ang Xu [*] Jiazhen Jing [*] Dinesha Dissanayake [*]

Abstract

This project aims to compare the performance of two supervised learning methods for the classification of six jellyfish species. Specifically, Convolutional Neural Network (CNN) and Transfer Learning based on the VGG16 architecture are evaluated in terms of accuracy, loss, and validation results. Our objective is to identify the method that exhibits superior performance in accurately classifying jellyfish species. Through an analysis of the models' performance metrics, we conclude that Transfer Learning outperforms the CNN method in achieving more precise classifications.

1. Introduction

In this study, we aim to develop multi-class classification models utilizing two different supervised learning techniques for categorizing six types of species of jellyfish commonly found in the ocean.

Accurately identifying these six species could assist in understanding marine ecosystems and their ecological dynamics. It contributes to a deeper understanding of how these species fit into marine food webs, contribute to nutrient cycling, and overall, maintain the health of the ecosystem.

Moreover, precise classification supports in identifying endangered or invasive species, thereby enabling the effective management of ecosystems to protect biodiversity.

The six jellyfish species are:

1. **Moon jellyfish (Aurelia aurita)**: Common jellyfish with four horseshoe-shaped gonads visible through the top of its translucent bell. It feeds by collecting medusae, plankton, and mollusks with its tentacles.
2. **Barrel jellyfish (Rhizostoma pulmo)**: The largest jellyfish found in British waters, with a bell that can grow up to 90 cm in diameter. It feeds on plankton and small fish by catching them in its tentacles.
3. **Blue jellyfish (Cyanea lamarckii)**: Large jellyfish that can grow up to 30 cm in diameter. It feeds on plankton and small fish by catching them in its tentacles.
4. **Compass jellyfish (Chrysaora hysoscella)**: Named after the brown markings on its bell that resemble a compass rose. It feeds on plankton and small fish by catching them in its tentacles.
5. **Lion's mane jellyfish (Cyanea capillata)**: The largest jellyfish in the world, with a bell that can grow up to 2 meters in diameter and tentacles that can reach up to 30 meters in length. It feeds on plankton and small fish by catching them in its tentacles.
6. **Mauve stinger (Pelagia noctiluca)**: Small jellyfish with long tentacles and warty structures on its bell full of stinging cells. It feeds on other small jellyfish and oceanic sea squirts.

2. Methods and Data

This data has been taken from Kaggle and contains about 979 images of jellyfish belonging to six different species (categories): mauve stinger jellyfish, moon jellyfish, barrel jellyfish, blue jellyfish, compass jellyfish, and lion's mane jellyfish. For each species, we have about 150 images, with a total of 900 images in the training set, 39 images in the validation set, and 40 images in the test set. Each image is a 2D data, which contains colors and the size of the images is $(224 \times 224 \times 3)$.

This study uses two distinct supervised learning methods to classify the jellyfish images which are Convolutional Neural Network (CNN) and transfer learning based on the VGG16 architecture.

2.1. Convolutional Neural Network (CNN)

A Convolutional Neural Network (CNN) is a type of neural network that is commonly used for image and video processing tasks. It is designed to process grid-like data, such as an image, by moving small filters across the data to detect patterns or features.

A CNN consists of several layers, including convolutional layers, pooling layers, and fully connected layers. The convolutional layers are responsible for extracting features from the input data, such as edges, corners, and shapes. The pooling layers reduce the spatial dimensions of the data to reduce the number of parameters and the number of computations. Finally, the fully connected layers are used for classification or regression tasks.

2.2. Transfer Learning based on VGG16

Transfer learning based on VGG16 is a technique used in deep learning where a pre-trained neural network (in our case, VGG16) is used to extract features from images. The idea behind transfer learning is that the pre-trained model has already learned to recognize certain features, such as edges, corners, and shapes, that are relevant to the new task, so it can be fine-tuned to perform well on the new task with less data and computational resources.

VGG16 is a popular Convolutional Neural Network (CNN) architecture that has been pre-trained on a large dataset (ImageNet) and is used here as a feature extractor. The VGG16 model consists of 13 convolutional layers, 5 max-pooling layers, and 3 fully connected layers. By employing VGG16, we can recognize complex patterns, thus helping in classifying jellyfish images despite the relatively limited size of our dataset.

3. Model Selection

For the training and evaluation of our models, the following parameter settings were used across the coding phase.

- Optimizer: Adam optimizer with a learning rate of 0.001
- Steps per Epoch: 40
- Epochs: 20,30
- Validation Steps: 1,3

3.1. CNN

The CNN model was employed following a data preprocessing phase. We initially trained a model using the training data generated through `flow_images_from_directory`. At this stage, the images were resized uniformly to dimensions (224, 224). During the model feeding process, batches of 15 images were introduced into the neural network. The class mode `sparse` was used which indicates that the labels are represented as integers since we are working with a multi-class task.

Following this, in a subsequent model, we utilized data augmentation to address the overfitting issue. So there, the input images went through various transformations using the **`image_data_generator()`**. Specifically, the pixel values were normalized to the [0, 1] interval.

3.1.1. Model 1

Our initial CNN model was implemented with three convolutional layers, three max-pooling layers, one flattened layer, and two dense layers. There is more detailed information.

For the Convolutional Layer, we did below settings:

- The first convolution layer with 16 filters has a kernel size of (3, 3) and padding. Further, the input shape is set as (224, 224, 3) to represent a standard RGB image.
- The second and third convolution layers with 32 filters have a kernel size of (3, 3).
- **ReLu** is applied as the activation function for all three convolution layers.
- A max-pooling layer with a pool size of (2, 2) is added after each convolution layer.

Then, a flattening layer is added to flatten the 3D tensor output to a 1D tensor for input to the dense layers.

Next, two fully connected dense layer with 32 units with **ReLU** and 6 units with **softmax** are added which are commonly used for multi-class classification.
This is a simple model, intended to build upon in subsequent models.

3.1.2. Model 2

Based on the results of Model 1, it fitted the data well, as accuracy reached almost 1. However the validation accuracy is low, and this implies an overfitting problem. In order to address this issue, we added image augmentation (Wiafe) for the second model which is a common method to avoid overfitting by tweaking the images a bit by applying random transformations such as rotation, cropping, zooming, etc. Other than that, the rest settings of Model 2 remain the same as Model 1.
The following transformations have been incorporated into the training data using the `image_data_generator`:

- **Rescaling:** Images have been rescaled by a factor of 1/255 to normalize pixel value.
- **Rotation:** Random rotate images in the range of -40 to +40 degrees.
- **Width Shift:** Random horizontal shift by a fraction of 0.2.
- **Height Shift:** Random vertical shift by a fraction of 0.2.
- **Shear:** Random shearing by a fraction of 0.2.
- **Zoom:** Random zooming by a factor of 0.2.
- **Horizontal Flip:** Random horizontal flipping.
- **Fill Mode:** Nearest neighbor fill mode for handling newly created pixels.

Such transformations help to expose the model to more aspects of the data and helping in better generalization of the model.

3.1.3. Model 3

In this case, we aim to reduce the overfitting by adding dropout. Dropout works by randomly removing nodes along with all their incoming and outgoing connections from a neural network during training. Here we compare the output of 0.2, 0.3, 0.4, 0.5.
We added a dropout layer with a rate of 0.5 after each con-

volution layer.

3.1.4. Model 4

We added one convolution layer with 32 filters with a kernel of size (3, 3) before the flatten layer. Further, we enlarged the filters for the first three convolution layers from (16, 32, 32) to (32, 64, 128) and increased the units of the fully connected dense layer from 32 to 64 to reduce the parameter number.

3.1.5. Model 5

However, adding one more layer and enlarging filters does not work for our model based on the results. In this step, we look back at model3 and try to change the kernel size for the second and the third convolution layer from (3, 3) to (4, 4).

3.1.6. Summary of the selected CNN model

The Keras model output of the selected CNN model (model 3) is shown below.

Layer	output shape	param
conv2d	(224, 224, 16)	448
dropout	(224, 224, 16)	0
max pooling2d	(112, 112, 16)	0
conv2d	(110, 110, 32)	4640
dropout	(110, 110, 32)	0
max pooling2d	(55, 55, 32)	0
conv2d	(55, 55, 32)	9248
dropout	(55, 55, 32)	0
max pooling2d	(27, 27, 32)	0
flatten	23328	0
dense	32	746528
dense	6	198
Total params: 761062		

Table 1. Structure of the selected CNN model

3.2. Transfer Learning

In contrast to the previously described CNN approach, our transfer learning method involved applying the VGG16 architecture pre-trained on the ImageNet dataset. This facilitates us to make use of learned features from a broader dataset, assisting in classifying our jellyfish images even though our dataset size is small.

3.2.1. Model 1

The VGG16 model was initialized using pre-trained weights from ImageNet and constructed with an input shape of (224, 224, 3).

The subsequent layers in the model architecture included a flattening layer and two dense layers.

The flattening layer transformed the output from the VGG16 model to a one-dimensional vector. Following the flattening layer, there were two dense layers; The first dense layer contained 64 units and used the ReLU activation function. The final dense layer contained 6 units and used the sigmoid activation function.

The weights of the pre-trained VGG16 layers were frozen to maintain the learned representations from pre-trained model on the new model to avoid overfitting and to reduce training time.

3.2.2. Model 2

In model 2, one modification was made to the VGG16 base. We froze the weights of the layers up to the **`block5_conv1`** layer. This act restricts further training to the initial layers.

3.2.3. Model 3

Model 3 is build on model1, we further unfreezing the VGG16 layers `block5_conv1` layer. This allowed for retraining and fine-tuning of these unfrozen layers and the subsequent layers during the training process.

3.2.4. Summary of the chosen Transfer Learning Model

Based on the results, we have chosen the third model as the best model (Discussed under the results section).

The Keras model output of the chosen VGG16 based Transfer Learning model is as follows.

Layer	output shape	param	Trainable
vgg16	(7, 7, 512)	14714688	Y
flatten	25088	0	Y
dense	64	1605696	Y
dense	6	390	Y
Total params:	16320774		

Table 2. Structure of the selected VGG16 model

4. Results

In this section, the findings of the study are thoroughly discussed.
In Table 3 below, we summarized the four matrices; training accuracy, training error, validation accuracy, and validation error of all the fitted models.

4.1. CNN

The accuracy of the CNN models can reach 1 and the corresponding training loss is not high for all five models. However, if we look at the validation accuracy and validation loss, we can find some problems. The validation loss is extremely high and the validation accuracy is less than 0.5. This result shows us a very typical overfitting case.

After we applied the data augmentation in model 2, the metrics improved compared to the initial model, although the validation accuracy did not improve much, the validation loss decreased from 7.2736 to 4.3911, indicating that the data augmentation pair reduced the overfitting problem in the initial model.

But from the metrics of model 2, we can see overfitting is

	Convolutional Neural Network					Transfer Learning - VGG16		
	Model 1	Model 2	Model 3	Model 4	Model 5	Model 1	Model 2	Model 3
Training Error	0.0117	0.0273	0.0039	0.3218	0.1422	0.3831	0.1019	0.0249
Validation Error	7.2736	4.3911	1.270	1.608	1.625	1.088	1.232	1.491
Training Accuracy	1.000	0.99	1.000	0.9033	0.9683	0.87	0.9733	0.9967
Validation Accuracy	0.450	0.5167	0.6667	0.3667	0.450	0.7167	0.80	0.8333

Table 3. Summary of all the fitted models

still not completely solved, in model 3 we added a dropout layer, which is also a common method we can use. Regarding the choice of hyperparameters, we tried four values and finally chose 0.5 as the dropout rate since it gives the best model performance among the four values. Now, model 3 with both data augmentation and dropout implemented can beat some overfitting problems but not all, but at least we know that is a step in the right direction.

Next, we need to continue to try to mitigate the overfitting problem and improve validation accuracy. We plan to tuning the hyperparameters such as the number of filters per convolution layer, or the number of layers in the network to get a better validation accuracy. Hence, in model 4 we adjusted the model accordingly, added one convolution layer, increased the number of filters, and finally added the hidden units. Unfortunately, model 4 does not give a better performance, but at least it is smooth without big floats.

Since model 4 did not give a better performance, we now revert to the structure of model 3. We try to change the kernel size of convolution layers to see if it allows model 5 to give the desired result. Once again, it did not appear to have the performance we wanted. To sum up, model 3 is

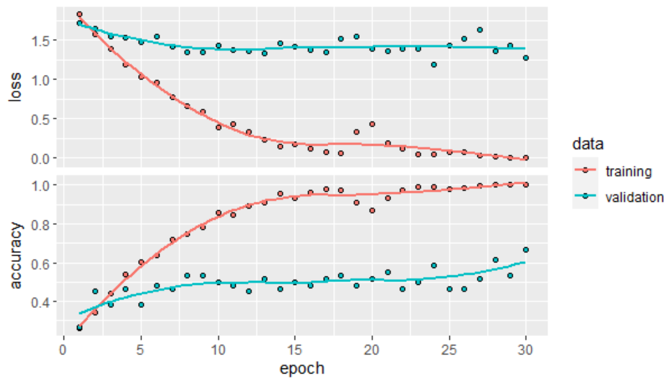

Figure 1. Loss and Accuracy of CNN Model

the best performing of our existing CNN models.

4.2. Transfer Learning

According to Table 3, we can see that Transfer learning models have a comparatively better performance than CNN models. In model 3 of Transfer learning, though it has a little bit higher validation error, it has the lowest training error, highest training accuracy, and highest validation accuracy. Hence, we conclude it as the best model under Transfer learning based on VGG16.

Moreover, the accuracy and Loss per Epoch for the final Transfer Learning model are shown in Figure 2. According to that, the training data accuracy shows roughly a straight line throughout the epochs, maintaining a high accuracy level and eventually reaching 99% after the final epoch. In contrast, the validation accuracy seems to start from around 0.75 and level off at around 0.83, indicating an improvement and at the same time indicating a noticeable discrepancy compared to the training accuracy. This discrepancy between the two accuracy metrics could be due to the persistent presence of over-fitting, where the model struggles to generalize to new data points in the validation set.

But this method is far better than the CNN method as we could achieve at least a 0.83 validation accuracy. Because in CNN we could only achieve only up-to around 0.67 accuracy.

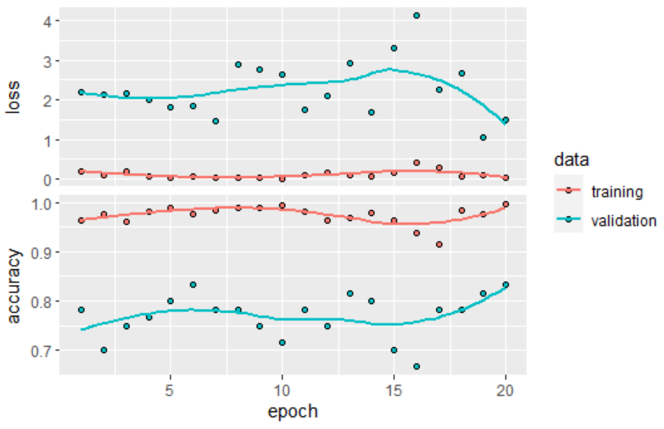

Figure 2. Loss and Accuracy of Transfer Learning Model

5. Discussion

Despite the discrepancy, the Transfer Learning method demonstrates an improvement compared to the CNN approach. Achieving a validation accuracy of at least 0.83 surpasses the performance of the CNN method, which reached only around 0.67. This improvement suggests that the second method, despite its limitations, performs more effectively than the CNN method in generalizing to unseen data.

The effectiveness of VGG16 might be due to its prior training on a diverse dataset, and it allows to identify patterns that are relevant to our particular domain.

For future work, it is beneficial to explore potential overfitting issues and strategies to improve the model's generalization capability on unseen data.

6. Conclusion

In conclusion, our investigation revealed that transfer learning based on VGG16 outperformed the CNN approach for our specific task. If we have small dataset, overfitting will be the main issue and data augmentation is a good way to deal with overfitting. Also transfer learning is better with small dataset.

References

E. Wiafe. Image augmentation and overfitting: An r version. URL https://rpubs.com/eR_ic/mlr_7.