SHIP TYPE CLASSIFICATION USING CNN

Abdülkerim Mustafa DEMİR
Istanbul Technical University
Istanbul / TURKEY
demira20@itu.edu.tr

***Abstract*—This report describes the building of a CNN model that estimates the class of ships using dataset that consist of ships. In addition, the effects of the hyperparameters that we use in convolutional neural networks, on the success of the model is evaluated. Also, using pre-trained models, their success on the dataset has been evaluated.**

I. INTRODUCTION

The development of technology has brought many innovations that make our lives easier. Many processes, which are long and hard to be done by humans, are being performed by computers and machines in a much faster and more practical way. With the improvements in convolutional neural networks, the success in applications such as object detection, object tracking, object classification, segmentation has increased. These topics are computer vision work spaces. In this project, we will examine the classification of different types of ships by using convolutional neural networks.

The types and characteristic structures of the ships are different in terms of their intended use. For example, cruise ships, commercial ships, underwater research ships and etc. Countries must be aware of maritime traffic in order to follow the research in the exclusive economic zone and to protect their maritime relevance and interests. Since territorial waters also a part of the countries borders, it is important for countries to identify ships entering their own areas. Determining the types of ships is also important for maritime activities such as maritime safety, maritime trade, fisheries management, combating marine pollution, and controlling the strait traffic. For example, the pollution generated by an oil-carring ship is not the same as that of cruise ship.

The aim of this project is to divide the ships in the data set into 5 classes according to their types. Using our training set, we will create a suitable model for classification and evaluate the success of our model. After building our model, we will estimate the classes of the unlabeled ships in the test set.

II. CNN OVERVIEW

Convolutional Neural Networks are a specialized version of artificial neural networks that enables the application of artificial neural networks on images and videos. CNN is a deep learning algorithm. It is successfully applied in computer vision areas such as detection, classification, segmentation. It captures the features in the images with different operations on different layers.

In this project, classification of ship types has been made. The image is made ready for classification by applying various processes to the image passing through Convolutional Layer, Pooling and Fully Connected Layers.

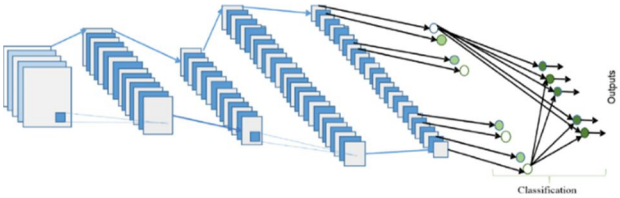

Fig. 1 : A visual showing the CNN architecture and layers

III. DATASET

Our data set contains 8932 ship images belonging to 5 different classes. 6252 of them are training set and 2680 of them are test sets. Ship images are provided in a directory. All images are JPG format. The dimensions of the ship images are not fixed. The names and classes of the ships belonging to the training set are given in a '.csv' file. The classes to which the ships belong are represented by numbers from 1 to 5. The ship types represented by the numbers are as follows.

1	Cargo
2	Military
3	Carrier
4	Cruise
5	Tankers

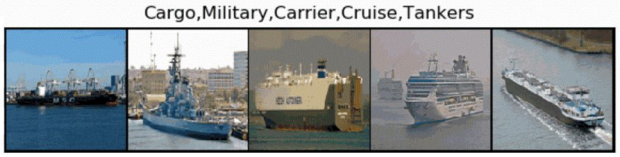

Fig. 2 : Sample ship images and classes from dataset

The distribution of classes in the dataset is not balanced. The number of Cargo class ships is more than other ship classes. Not having a balance in our dataset may result in the model being biased. For this reason, data augmentation process was applied to increase the size of the dataset and to ensure a balanced distribution of classes.

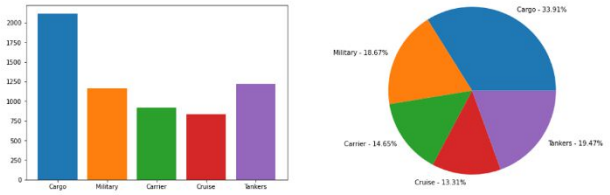

Fig. 3 : Class distributions in the training set (Cargo:2120, Military:1167, Carrier:916, Cruise:832, Tankers:1217)

The names of the pictures in the test set are given in a separate cvs file. The labels of the images are not provided in this file. That's why, by creating a validation set in our train dataset, we will evaluate the success of our model on validation set. We will assign the labels of the images in the test set with the model we built.

IV. DATA AUGMENTATION

It was stated above that the unbalanced distribution in the classes of the data set will cause model bias. We don't want our model to be biased. Therefore, by applying data augmentation to our dataset, we will increase the samples in our dataset and we will ensure that the classes are balanced distribution.

Data augmentation process enables data to be increased with various techniques. For pictures, rotation, translation, clipping, blurring, adding noise, flipping, etc can be used in data augmentation. For voices; frequency masking, scaling and for texts; methods such as replacing with synonyms, deleting random words can be used in data augmentation. In this project, rotation, flipping, mirroring, blurring and unsharp masking processes were applied to the images in the training set.

Fig. 4 : Obtained images by applying blurring and unsharp masking to the original image

The images obtained from these operations were saved on the computer. Then, the images belonging to the corresponding classes according to their distribution in the dataset were included in the directory belonging to the dataset. A new '.csv' file has been created with the image names and labels of these images.

As a result of these operations, we have reached a training set consisting of 4000 ship images belonging to each class. The total number of images in our training set increased from 6252 to 20000.

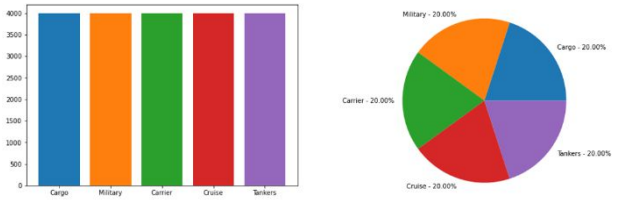

Fig. 5 : Class distributions in the training set after data augmentation (4000 images of each class)

V. TRAINING SET / VALIDATION SET SPLIT

After the Data Augmentation part, some of the data in the training dataset was separated as the validation set. Thus, we will be able to evaluate the performance of CNN models on the validation set. We will adjust the layers and hyperparameters according to the results we get. We splitted 20 percent of the training set into a validation set. So, we have a training set consisting of 16000 images and a validation set consisting of 4000 images.

VI. MODEL ARCHITECTURE

Now, we will firstly create our CNN model. A VGG16-based model was created as the initial model. The input of the model is the ship image and the output is 5 different ship classes. Since we use supervised learning, weights will be updated at every step of the training and we will try to find the optimum weights.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 128, 128, 64)      1792
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 128, 128, 64)      36928
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 64, 64, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 64, 64, 128)       73856
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 64, 64, 128)       147584
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 32, 32, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 32, 32, 256)       295168
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 32, 32, 256)       590080
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 32, 32, 256)       590080
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 16, 16, 256)       0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 16, 16, 512)       1180160
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 16, 16, 512)       2359808
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 16, 16, 512)       2359808
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 8, 8, 512)         0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 8, 8, 512)         2359808
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 8, 8, 512)         2359808
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 8, 8, 512)         2359808
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 4, 4, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0
_________________________________________________________________
dense (Dense)                (None, 4096)              33558528
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 20485
=================================================================
Total params: 65,075,013
Trainable params: 65,075,013
Non-trainable params: 0
_________________________________________________________________
```

Fig. 6 : Baseline Model for Ship Type Classification

Since the dimensions of the images in our dataset are not fixed, we resize the images as 128x128x3 while we import the images into our working environment. We also normalize the pixel values of the images. Thus, pixel values between 0 and 255 will be set between 0 and 1.

Initially, we set the batch size to 32. So the model will run on 32 training samples before the weights are updated. The GPU was used while training the model. GPUs are much faster compared to CPUs and have an important place in convolutional neural networks. Using the GPU, CNN models can be trained quickly. Each epoch in our baseline model took about 59 seconds. 'Adam' was chosen as the optimizer parameter. Learning rate was set to '0.00001'. The number of epoch was also set to 25. At the end of the training with this model and hyperparameters, we achieved training accuracy as 98.79% and validation accuracy as 83.99%.

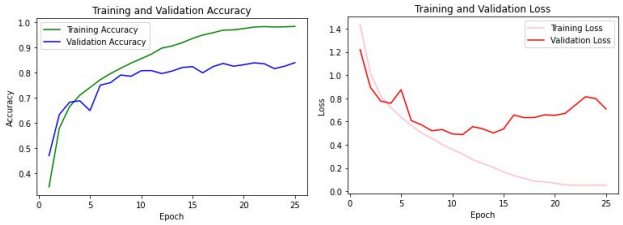

Fig. 7 : Training & Validation Accuracy – Trainig & Validation Loss for Baseline Model

VII. OPTIMIZATION

As far as our initial training result concerned, we achieved a 98.79% train accuracy, while we achieved a validation accuracy as 83.99%. We will try to get better results on the validation set by tuning the hyperparameters correctly and we will observe the effects of hyperparameters.

A. INITIALIZER

The models in this project were created using the keras library. Layers in Keras library start with 'Xavier initializer' by default. "Xavier initializer is the weights initialization technique that tries to make the variance of the outputs of a layer to be equal to the variance of its inputs." Xavier initialization works better for layers with sigmoid activation. Since we use the ReLU function as the activation function in the model, He initialization was used as the initializer. Because, 'He initialization' works better for layers with ReLU activation.

When the model was started with the he-uniform initializer, training accuracy was achieved as 99.69% and validation accuracy was achieved as 85.32%.

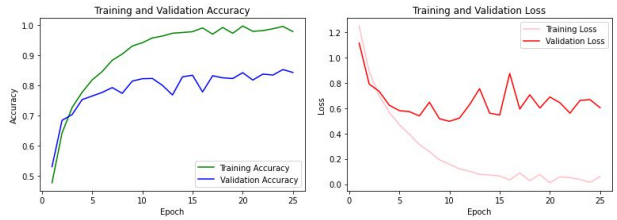

Fig. 8 : Training & Validation Accuracy – Trainig & Validation Loss after 'he_uniform' initializer

B. OPTIMIZER

One of the most commonly used optimizers to train models in deep learning is Adam. In this project, we used Adam as an optimizer while compiling our models. In order to compare Adam's success, we compiled the model that we created last by choosing SGD as the optimizer. At the end of the training with the same learning rate, batch size and number of epochs, we got the training accuracy as 75.65% and the validation accuracy as 70.57%. As you can see, we got very low accuracy compared to Adam optimizer. Therefore, we used Adam as the optimizer while compiling the models that follow the project.

C. REGULARIZATION

In the model trainings until now, 99.69% accuracy in training set and 85.32% accuracy in validation set have been achieved. To achieve a higher accuracy in the validation set, we need to apply other methods. One of these methods is regularization to generalize the model.

1) Dropout

On each training iteration, a dropout layer randomly removes all inbound and outbound connections, as well as some nodes in the network according to the specified value. Dropout can be applied to hidden layera or input layer. The most commonly used dropout value is 0.5. So, dropout value was chosen as 0.5 in the first training using dropout. Then dropout was applied to the fully connected layers by selecting values of 0.4 and 0.2.

	Dropout (Fully Connected Layers)	Training Accuracy	Validation Accuracy
Trial 1	0.5	98.17%	86.25%
Trial 2	0.4	98.87%	85.77%
Trial 3	0.2	99.01%	89.87%

Table 1 : The effect of dropout applied to fully connected layers on training and validation accuracy

As can be seen from the table, when '0.2' dropout was applied to fully connected layers, we achieved validation accuracy as 89.87%.

2) L2 Regularization

L1 and L2 regularization are the most common regularizaition types used in convolutional neural networks. They update the overall cost function by adding another term called regularization term. Because of the addition of this regularization term, the values of the weight matrices decrease.

$$Cost\,function = Loss + \frac{\lambda}{2m} * \sum \|w\|^2$$

In this formula, lambda is the regularization parameter. It is the hyperparameter whose value is optimized for better results. We applied L2 regularization by giving different parameters to fully connected layers. After applying L2 regularization to fully connected layers, the accuracies in training and validation sets are as in the table.

	L2 Regulazition (Fully Connected Layers)	Training Accuracy	Validation Accuracy
Trial 1	0.001	98.99%	90.10%
Trial 2	0.0001	99.39%	90.89%
Trial 3	0.00001	99.01%	90.00%

Table 2 : The effect of L2 Regularization applied to fully connected layers on training and validation accuracy

As can be seen from the table, when '0.0001' L2 Regularization was applied to fully connected layers, we achieved training accuracy as 99.39% and validation accuracy as 90.89%.

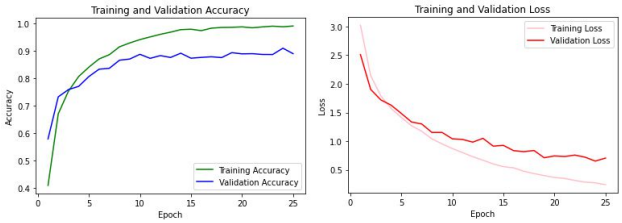

Fig. 8 : Training & Validation Accuracy – Trainig & Validation Loss after L2 Regularization (According to 0.0001)

D. NORMALIZATION

Some features may have higher numerical values compared to other properties. This may cause the feature with a high numerical value to suppress other features and cause the model to be biased. Normalization methods normalize properties to maintain the contribution of all properties.

1) Batch Normalization

Batch normalization is a normalization method that normalizes activations in a network across the mini-batch. "For each feature, batch normalization computes the mean and variance of that feature in the mini-batch. It then subtracts the mean and divides the feature by its mini-batch standard deviation."

When batch normalization process was applied on convolutional layers and fully connected layers, training accuracy was 98.51% and validation accuracy was 87.35%.

2) Layer Normalization

"Layer normalization normalizes input across the features instead of normalizing input features across the batch dimension in batch normalization." When layer normalization process was applied training accuracy was 99.51% and validation accuracy was 89.49%.

E. NUMBER OF EPOCHS

The number of epochs is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset. In previous trials the number of epoch was set to 25. By increasing the number of epochs, we will observe the effect of the model on its success.

	Number of Epochs	Training Accuracy	Validation Accuracy
Trial 1	25	99.39%	90.89%
Trial 2	50	99.67%	90.72%
Trial 3	100	100%	91.60%

Table 3 : The effect of number of epoch on training accuracy and validation accuracy

When we set the number of epochs to 100, we achieved 100% accuracy in training set and 91.60% accuracy in validation set.

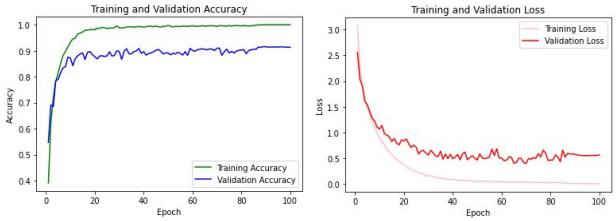

Fig. 9 : Training & Validation Accuracy – Trainig & Validation Loss after 100 number of epoch

Many trials have been made to further increase the success of the model. Additional layers was added to fully connected layers. Average Pooling was applied before passing from convolutional layers to fully connected layers. Then, batch normalization was applied to convolutional layers and fully connected layers after activation function. When the built model was trained in 100 epochs, its validation accuracy reached 92.97%.

VIII. PRE-TRAINED MODELS

We have observed that the correctly tuning hyperparameters are important to the success of the model. By increasing the depth of the model and working on hyperparameters, the generalizable and success of the model can reach better levels. We will now observe training and validation accuracies using pre-trained models in our dataset. It is a long and tiring process to train a CNN model from scratch and adjust its hyperparameters. These models, that we will use, have been pre-trained and their weight is recorded. According to the results we get using the recorded weights in these models, we can further increase the accuracy we get by tuning the hyperpameters or deepening the ready-made model. This method is also called Transfer Learning.

Models	Pool	E	LR	Training Accuracy	Validation Accuracy
ResNet50	max	50	0.000001	99.77%	86.47%
ResNet101	avg	25	0.000001	99.34%	92.57%
ResNet101	max	50	0.000001	99.89%	88.55%
ResNet50	avg	50	0.000001	99.88%	93.62%
VGG16	avg	50	0.000001	100%	93.19%
Xception	max	25	0.0001	99.80%	96.35%
Xception	avg	25	0.0001	99.88%	97.00%

Table 4 : Results obtained with different pre-trained models

The pooling layer in Table 4 was applied just before switching to fully connected layers. In the Xception pre-training model, we reached 97% in validation accuracy using the hypreparameters in Table 4.

IX. EVALUATION OF THE MODEL

In the model we developed starting from the baseline, we reached 92.97% validation accuracy. We can see the prediction distribution of the classes in the confusion matrix of this model. When we examine the confusion matrix, we can see that cargo and tanker class ships are more confused by the model compared to other classes. (Prediction is cargo but correct class is tanker or prediction is tanker but correct class is cargo.)

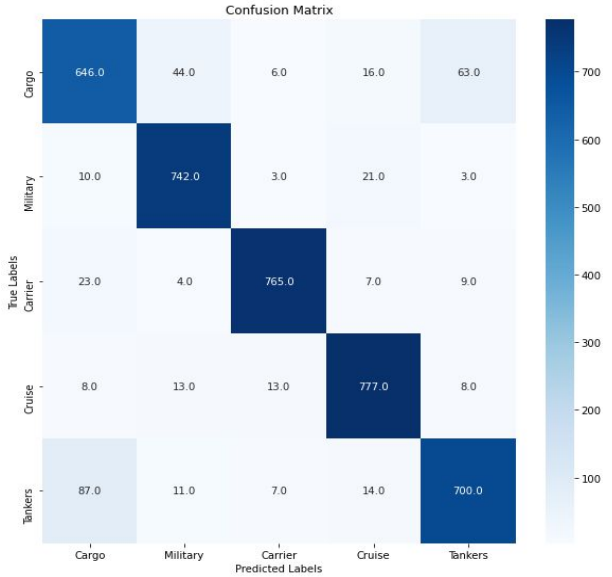

Fig. 10 : Confusion Matrix

X. CONCLUSION

In this project, we built a classification model that classifies the ships in the dataset according to 5 different classes. We observed the effect of hyperparameters on the success of the model. We tried to find the optimum values by tuning the hyperparameters. We have observed that correctly tuned hyperparameters increase the success of the model. In addition, in many trials, it was observed that the increase in model depth positively reflected on the success of the model. Because, deep neural networks learn more discriminative features as the layers go deeper. As our model gets deeper, we can increase the number of epocs.

Another conclusion is that creating a CNN model from scratch is a tiring and long process. Therefore, we can do improvements according to our dataset by using pre-trained models.

REFERANCES

[1] https://stats.stackexchange.com/questions/319323/whats-the-difference-between-variance-scaling-initializer-and-xavier-initialize

[2] https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/

[3] https://towardsdatascience.com/backpropagation-and-batch-normalization-in-feedforward-neural-networks-explained-901fd6e5393e

[4] https://medium.com/techspace-usict/normalization-techniques-in-deep-neural-networks-9121bf100d8

[5] arXiv:1311.2901v3 **[cs.CV]**

[6] https://www.learnopencv.com/keras-tutorial-transfer-learning-using-pre-trained-models/

[7] arXiv:2011.06702v1 **[cs.LG]**

[8] http://cs231n.stanford.edu/