Artificial Neural Network

Note: Project 3

Xiangsheng Chai
ECE Department
UFID: 44036693

***Abstract*—In this project, artificial neural network (ANN) is used to train two models, flower species classification model and car detection model. In flower species classification model, three different kinds of models with ANN, with costumed CNN and with transfer learning are applied to find the best model. By experiment with several hyperparameters, the significance of each hyperparameter will be found. In the car detection model, we apply transfer learning to do the regression task of bounding boxes coordinates, and use different validation metrics to test the performance of model. Then, different cases like no car present in the image are considered. In this project, we will use TensorFlow to apply multiple ANN strategies. Finally, by applying different kinds of methods, we will draw a conclusion of this project.**

Keywords—Flower Species Dataset, Car Detection Dataset, TensorFlow, Artificial Neural Network, Transfer Learning.

I. Introduction

Artificial neural networks (ANNs), usually simply called neural networks (NNs) or neural nets, are computing systems inspired by the biological neural networks that constitute animal brains. ANNs have been around for quite a while: they were first introduced back in 1943 by the neurophysiologist Warren McCulloch and the mathematician Walter Pitts. They proposed a very simple model of the biological neuron, which later became known as an artificial neuron: it has one or more binary inputs and one binary output. McCulloch and Pitts showed that even with such a simplified model it is possible to build a network of artificial neurons that computes any logical proposition you want.

In this project, we will use TensorFlow to build our machine learning model. TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks. TensorFlow was developed by the Google Brain team for internal Google use in research and production. The initial version was released under the Apache License 2.0 in 2015. Google released the updated version of TensorFlow, named TensorFlow 2.0, in September 2019. For this project, I use TensorFlow 2.7 in hypergator. Keras is a high-level Deep Learning API that allows you to easily build, train, evaluate, and execute all sorts of neural networks. TensorFlow itself now comes bundled with its own Keras implementation. It only supports TensorFlow as the backend, but it has the advantage of offering some very useful extra features.

There are totally two datasets. Dataset 1 is flower species dataset. The dataset is saved as numpy array and contains a total of 1678 RGB images from 10 classes. The goal is to utilize the training images to train a flower species classifier, and make predictions for the test set. Each RGB image is of size 300×300×3. The sample's distribution of per classes is shown below:

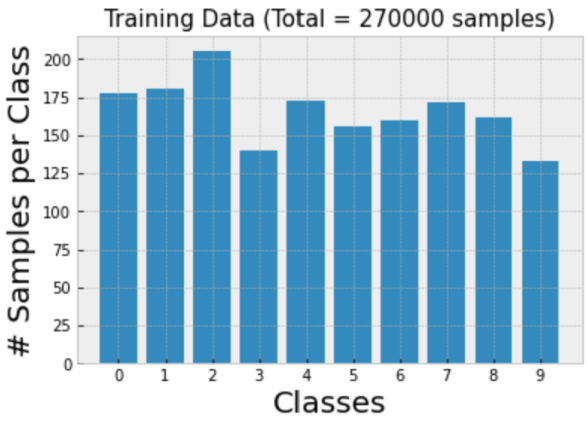

Fig. 1. Distribution of samples in each class.

Dataset 2 is car detection Dataset. This dataset contains labeled annotations for 559 training samples. Each annotation corresponds to a bounding box of the object car. A bounding box has totally 4 output values of target, which correspond to the top-left and bottom-right (x, y)-coordinates, respectively. The goal is to train an object (car) detection artificial neural network using the training samples, and make predictions for the images in test.

In this project, our goal is to implement artificial neural networks with TensorFlow, and there are totally 3 questions for us to solve. The questions are listed below:

1. Train an artificial neural network for flower species classification using the training set of dataset 1.

2. Discuss how you would validate performance in the test set given that no target labels are provided. Discuss also how you would address the case where no car is present in the image.
3. Train an artificial neural network for object detection using the training set of dataset 2.

II. Implementations

A. Preprocessing

Dataset 1 contains 1678 images from 10 classes in training. Each RBG image is of size 300×300×3 pixels. The original data has size of 1658×270000. We first split it into training dataset and validation dataset used for training neural network, then applying minmax scaler to scale each pixel between 0 and 1. The we use TensorFlow to reshape the image size to 300×300×3.

Dataset 2 contains 559 images with size of 380×676×3. These 559 images all contain at least one car in it. It has target with size of 559×4, which 559 corresponds to each image and 4 corresponds to the top-left and bottom-right (x, y)-coordinates. Firstly, we load the training images and targets in numpy arrays with size of 559×770640 and 559×4. Then minmax scaler is applied in both data and target in order to keep the value of targets. Finally, I selected first 50 data as my validation dataset and the others as the training dataset. This is because the training dataset contains lots of same image, we don't want the same image occurs in both train and validation set. In order to consider the case where no car present in the image, all training data should be used and add fix target label [0,0,0,0] to the image without the car.

B. Image classification task

In this task, our goal is to utilize the training images to train a flower species classification (an artificial neural network architecture), and make predictions for the test set. I applied totals three architecture in this task.

The first architecture is simple multi-layer perceptron (MLP). I didn't use any convolution layers in this model. Two hidden layers with 300 units and 100 units respectively were applied, and the output layer contains total ten neurons with softmax activation function. 10 neurons in output layer represent the 10 classes. MLP flatten the size of 300×300×3 images into size of 270000 array, and it will cause the spatial information loss.

The second architecture is costumed convolution neural network. It contains three convolution layers, three pooling layers and two hidden layers. Two hidden layers have the same number of units as the first model. The convolution layer contains 64,128 and 256 neurons respectively.

The third architecture is applied with transfer learning. The pre-trained model I used is Xception in Keras applications. Xception was proposed in 2016 by François Chollet, and it has a special type of layer called a depth-wise separable convolution layer. It is a powerful model which is pre-trained by image net. We only apply it without the top layer, and add an output layer with 10 neurons in order to adapt our flower classification task. Obsoletely, transfer learning model has the most complicated architecture, and it will perform the best. Then, we will tune the batch size and learning rate in order to see the influence of these hyperparameters.

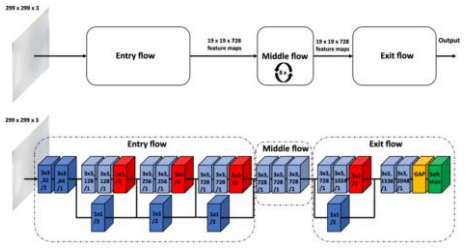

Fig. 2. Structure of Xception.

C. Object detection task

In this task, our goal is to train an object (car) detection artificial neural network using the training samples, and make predictions for the images in test. Transfer learning with pre-trained model VGG16 was applied in this problem. VGG16 is a convolutional neural network for classification and detection. The structure of VGG16 is shown below

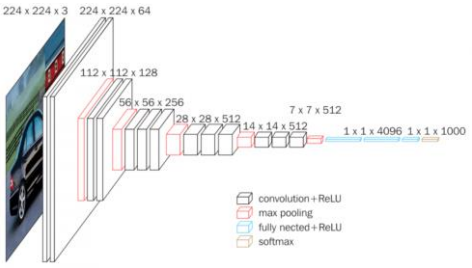

Fig. 3. Structure of VGG16.

VGG16 has a very simple and classical architecture, and it is powerful in detection. However, its simple architecture causes more parameters to train and more spaces for saving model. First, we only consider the case that there is at least one car present in the image.

Firstly, I grab the dimensions and scale the bounding box coordinates to the range [0, 1]. The reason why I scale the bounding box is that the targets' values are pixel based. If we resize the image, the target should be changed. In this case, it is better to scaler the target into the range [0, 1]. After that, we load VGG16 with pre-trained ImageNet weights, chopping off the old fully-connected classification layer head and freeze all layers in the pre-trained model. Then, I performed network surgery by constructing a new fully-connected layer head that will output four values corresponding to the top-left and bottom-right bounding box coordinates of an object in an image.

After training, performance in the test set was reported in quantitative measure and qualitive measure. For qualitative measure, we will select ten images contained cars in the test set and set our own test label by MakeSenseAI, then visualize the result. For quantitative metric, overlapping region of interest (ROI) was applied. ROI calculates the overlapping region between test bounding box and predict bounding box divided by area of test bounding box. The accuracy score is between 0 and 1, and when the accuracy score is 1.0, it represents an exact

match. In practice, an accuracy score below 0.5 is probably poor match.

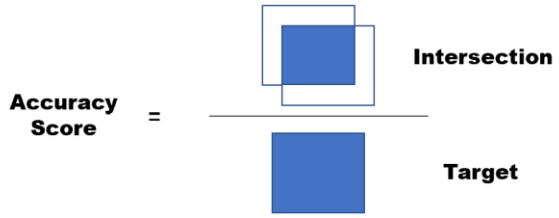

Fig. 4. Region of Interest.

III. Experiment

A. *Problem 1*

For problem 1, I first use a simple MLP to build the architecture. Model 1 contains two hidden layers with 300 and 100 neurons respectively, and output layer has 10 units with softmax activation function. Model 2 use convolution neural network. Three convolution layers with pooling layers are added above the MLP model. Model 3 use transfer learning strategy, and pre-trained model Xception was used. The top layer of Xception was replaced with an output layer with 10 units, applied by softmax activation function. The training performance of three models are shown below:

TABLE I. Three models' Performance

	Accuracy In training	Accuracy In validation
MODEL 1	1.00	0.54
MODEL 2	0.96	0.62
MODEL 3	1.00	0.88

From the result above, we can see the first model is apparently overfitting, but its performance in validation is not very well. This is because MLP flatten the images into size of 270000, and it causes the loss of spatial information. Model 2 is also overfitting, and it has a better performance in validation set than model 1. However, the convolution architecture is not sufficient to extract the specific features. Model 3 is the best, it was chosen as the final model in problem 1, and hyperparameter tuning was applied.

Model 3 used the pre-trained model as bottom, and connected with an output layer only. I tried to add a hidden layer between the pre-trained model and output layer, it surely saves the time to converge, but it didn't make any contribution on improving accuracy in validation. And, I set the early stop callback and epoch could not be considered as a hyperparameter. Hence, the main hyperparameters we need to tune is the batch size and the learning rate. In problem 1, I used the Nadam optimizer for all models. Firstly, I keep the learning rate 0.001 and change the batch size. I applied three types of batch size, online learning, mini batch with 32 samples and whole samples as a batch. The learning curves are shown below.

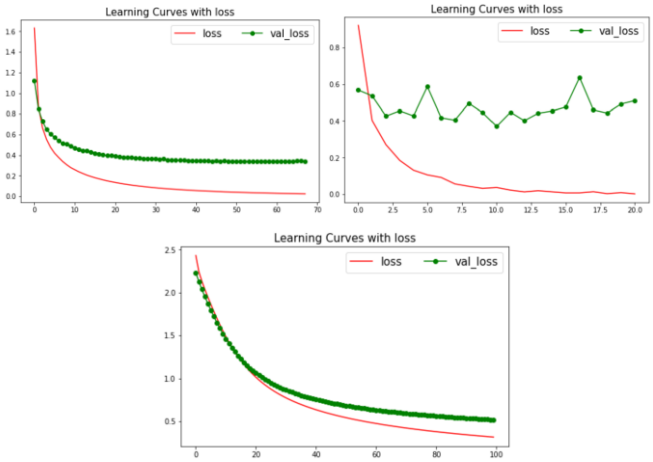

Fig. 5. Learning curves of mini batch learning (top left), online learning (top right) and batch learning (bottom).

We can see that online learning use the least epochs to converge, but its cross entropy in validation is unstable. Online learning uses the longest time in each epoch, so it doesn't save time on training. Since batch size learning use all samples as one batch to train, it takes the most epochs to converge. In this situation, 100 epochs are not sufficient for batch learning to converge. Compare to other two models, mini batch learning with 32 samples has the best performance. It converges relatively fast and has a better accuracy in validation set.

What's more, I keep the batch size at 32 and change the learning rate. I applied three values of learning rate, 0.001, 0.01 and 0.1. The learning curves of different learning rate are shown below.

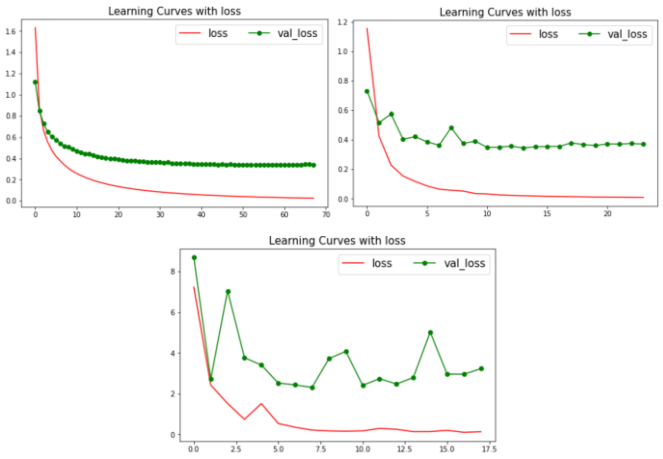

Fig. 6. Learning curves of 0.001 learning rate (top left), 0.01 learning rate (top right) and 0.1 learning rate (bottom).

From the result, model with 0.1 learning rate uses less epochs to drive to the relatively low cross entropy, but the validation cannot converge well. Model with 0.001 learning rate can converge to the result, and its accuracy perform best. Model with 0.01 learning rate use less time to converge than

model with 0.001 learning rate. In problem 1, training doesn't take too much time, hence I'd like to use 0.001 as my final model's learning rate.

Hence, my final model of problem 1 uses mini batch learning with 0.001 learning rate. After applying the test set, we got a relatively great accuracy 0.86 in test set.

B. Problem 2 & 3

First, I loaded the images and the target from the csv file. Then, I scale the images and the bounding box coordinates in target relative to the spatial dimensions of the input image. Some of images in the training data are the same. In order to avoid same images meet in both training set and validation set, I selected first fifty images as validation data instead of randomly splitting.

Then I used transfer learning with pre-trained model VGG16 and connected to a new fully-connected layer head that will output four values corresponding to the top-left and bottom-right bounding box coordinates of an object in an image to train. The learning curve during training epoch is shown below.

Fig. 7. Learning curve of problem 3.

Then we load the model in the test file, and select 10 images with own created labels in the test dataset. The labels were created by MakeSenseAI. The accuracy score of ROI in ten images is [0.51, 0.22, 0.61, 0.39, 0.23, 0.62, 0.56, 0.72, 0.40, 0], and the mean accuracy is 0.43. We will show some case in well performance and bad performance in below:

Fig. 8. Visualization of well performances in test set.

Fig. 9. Visualization of bad performances in test set.

From the visualization, we can see that car object in the margin of the image and in small size are hard to detect in my detection model. I also chose some images with multiple cars and the bounding boxes perform pretty bad. It tells us that only use CNN model could not perform well in such car detection. I found some essays on Internet, and lots of them recommended Yolo as the best approach to successfully dealing with car detection. Unfortunately, I didn't have enough time to learn the Yolo and apply it on this project.

Consider the case where no car is present in the images. I have two approaches to achieve it. First approach is to train two CNN models. One model does the image classification task to determine whether there is a car in the image, and if the determination is positive, then the other model will do the car detection work. The second approach is to train the images all together and set a certain bounding boxed' coordinates to the image without cars, for instance, set the label [0,0,0,0] for all images without a car.

IV. Conclusion

In this project, we got a high accuracy in flower classification task but a relatively poor performance in car detection task. We can see the advantages of transfer learning, and there still some approached we can use in detection task like Yolo.

Artificial Neural Network is a powerful tool to solve various problems in many areas. It can successfully achieve the goal of image classification task and object detection task. Transfer learning provide us lots of formidable pre-trained model which help us to save time in training and get better result. There are still many of applications of ANN, and I am thrilled to explore these in the future.