Project 3 - Artificial Neural Networks

Yuyang Zhang

Electrical and Computer Engineering, University of Florida

EEL 5934 Applied Machine Learning

Gainesville, U.S.

yuyangzhang@ufl.edu

*Abstract*— In this project, we trained different artificial neural network models to complete flower species classification tasks and car detection tasks. We will find the best model for both tasks by training multiple neural network models and testing their performance. In the car detection task, we will also discuss how to deal with abnormal situations in the training set. In this article we use TensorFlow to create the model. Through experiments, we found that transfer learning with Xception model performed well in both tasks and became our first choice when choosing a model. The transfer learning with Xception model for flower type classification can achieve very high performance metrics. But the transfer learning with VGG16 model for car detection performs bad, but it still stand out in comparison with other classic models.

# Introdunction

In this experiment we use artificial neural network (ANN) models to complete flower classification tasks and car detection tasks. An artificial neural network is composed of interconnected nodes, often called neurons or perceptions, organized into layers. The layers typically include an input layer, one or more hidden layers, and an output layer. The creation of ANN models is completed using TensorFlow. TensorFlow is an open-source machine learning framework developed by the Google Brain team. For the flower classification task, we created models using three common neural networks. After training and hyperparameter tuning, we compare their performance metrics and select the best model among them. For the car detection task, we select models based on empirical success and similarly adjust their hyperparameters and compare performance metrics to obtain the best model. Finally, there is a discussion about the no target label case and no car in the image case that may be encountered in the car detection task.

# Methodology

In this experiment, the types of artificial neural network models applied to the flower classification task are:

Artificial Neural Network (ANN) with 2 hidden layers - includes two layers of units between the input and output layers.

Convolutional Neural Network (CNN) - a type of ANN designed specifically for processing structured grid data by using convolutional layers and pooling layers.

Transfer Learning - involves leveraging pre-trained neural network models on one task and adapting them for a new, possibly related, task.

The artificial neural network model applied to the car detection task is Visual Geometry Group 16 (VGG16), a specific variant of the VGG series of convolutional neural networks (CNNs). It is often employed as a feature extractor in the early layers of the network, and its weights are fine-tuned on specific datasets for tasks such as image classification and object detection implementation steps.

We use TensorFlow 2.7.0 for all the tasks.

In this article, we need to complete the following tasks:

1. Implement ANN with 2 hidden layers, CNN and Transfer learning models for the flower classification task, adjust the hyperparameters and repeat the experiment to obtain the best hyperparameter combination for each model. Compare the performance metrics of three models and choose the best one.
2. Implement the VGG16 model for the car detection task, adjust the hyperparameters based on the performance indicators and output images, and repeat the experiment to obtain the best hyperparameter combination for the model. Discuss how would validate performance in the test set given that no target labels are provided. Discuss also how would address the case where no car is present in the image.

# Experiment

## Flower Classification

The dataset ‘Dataset 1’ for this task contains 10 kinds of flowers species images where the flower species are: Roses, Magnolias, Lilies, Sunflowers, Orchids, Marigold, Hibiscus, Firebush, Pentas, and Bougainvillea. The labels are 0-9 for corresponding species. Dataset 1 contains 1678 images from 10 classes in training. Each RBG image is of size 300x300x3 pixels. The original data has size of 1658x270000

1. *Preprocessing*

Start by splitting the data into training and validation sets. The split is done with a training/validation size of 80%/20%. Then we scales the pixel values of the image data to be between 0 and 1 by dividing each pixel value by 255.0. This normalization is common in image processing tasks and can help the model converge faster during training. Then we reshape the training and validation data into 4D tensors, where the dimensions correspond to the number of samples, image height, image width, and the number of color channels. Additionally, the data is converted into TensorFlow constant tensors with reduced precision. Using reduced precision can reduce memory usage and speed up computations.

1. *ANN with 2 hidden layers model*

This architecture is a simple multilayer perceptron (MLP). It has two hidden layers of 300 units and 100 units respectively, and the output layer contains a total of 10 neurons with softmax activation function. For both hidden layers we use a dropout of 0.2 to prevent overfitting. The 10 neurons in the output layer represent 10 categories. MLP flattens the size of a 300×300×3 image to the size of an array of 270000, which results in loss of spatial information.

For the weight I chose he\_normal. It initializes the weights with random values sampled from a normal distribution (Gaussian distribution) with a mean of 0 and a standard deviation calculated based on the number of input units in the weight tensor. This initialization method helps to mitigate the vanishing or exploding gradient problem that can occur during training.

With empirical success we use ReLU as the activation function of the hidden layer. This can help to solve vanishing gradient. For the output layer we choose Softmax because it can convert the original output of the model into a class probability distribution, since in this task we have 10 different labels.

After creating the model, we need to compile the model's learning process according to Keras. We specify the optimizer to be used during training as ‘Nadam’. Nadam is an optimizer that combines the benefits of Nesterov Accelerated Gradient (NAG) and Adam. It's a popular choice for training deep neural networks. For the loss function to be used during training we choose sparse categorical crossentropy. This is suitable for multi-class classification problems with integer labels. Metrics is used to specify the evaluation metrics to be monitored during training, in this case it's accuracy.

EarlyStopping is a callback that stops training the model when a monitored metric has stopped improving. Its monitoring metric is validation loss. Training was set to stop after 15 epochs when the monitoring metric showed no improvement. We also set up a callback that saves the model after every epoch with the best performance.

Finally, we set the number of training epochs for the model to 100 epochs and the training batch size to 32. These are derived from empirical successes. After many experiments we choose 0.00001 as the learning rate. Although this value is small, it does not affect the speed of the model and helps the model achieve higher validation accuracy. We fit the model and output its performance metrics. This is done through a method called 'Evaluate\_performance', which will output the classification report, accuracy and confusion matrix of the model using the training and validation sets. In addition, it also outputs the learning curves with loss and learning curves with accuracy generated by the model. The model can achieve an accuracy of 1.0 on the training set, and an accuracy of approximately 0.58 on the validation set. This means the model is overfitting and has a low accuracy on this task.

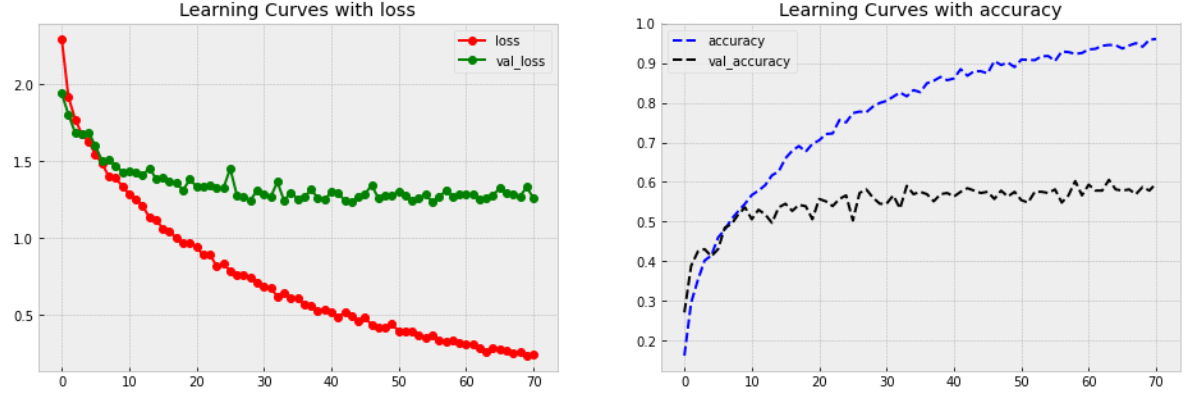


Fig. 1 Learning Curves with loss and accuracy for ANN with 2 hidden layers model

1. *CNN*

The second model is a custom CNN. It contains three convolutional layers, three pooling layers and two hidden layers. The two hidden layers have the same number of units as the first model.

The first convolutional layer has 64 filters, a kernel size of 10x10. The number of filters and kernel size are derived from empirical success. Same as ANN, we use ReLU to help in mitigating the vanishing gradient problem. Zero-padding helps in preserving the spatial information from the input to the output. The 'lecun\_normal ' initializer samples weights from a truncated normal distribution with a mean of 0 and a standard deviation that is proportional to the square root of the number of input units. In convolutional layers, the ‘lecun\_normal’ initializer helps in providing a good starting point for learning hierarchical features. In first pooling layer, the (2, 2) pool size is a common default in many CNN architectures.

Larger kernels in the initial layers can capture broad patterns, and smaller kernels in deeper layers can capture finer details. So we keep the same configuration in the second convolutional layer as the first layer except increase the number of filters to 128 and the kernel The size is reduced to 5x5. The second pooling layer is also the same as the first layer. Same idea, the third convolutional layer has 256 filters and the kernel size is 3x3. Then use flatten layer to flatten the 3D output to a 1D vector. The hidden layer and output layer below are the same as the previous ANN with 2 hidden layers model. All the layers described above make up our custom CNN model.

We use the same learning rate of 0.00001 as ANN with 2 hidden layers. This is also the result of multiple tests, and the reason is also because 0.00001 can achieve higher accuracy. Through the same compile and fit as ANN with 2 hidden layers (parameter settings are the same), we can obtain the performance of CNN. CNN can also achieve an accuracy of 1.0 under the training set. Under the validation set, an accuracy rate of approximately 0.73 can be achieved. From this we can know that the model still overfitting. But CNN has a higher accuracy on the validation set than ANN with 2 hidden layers.

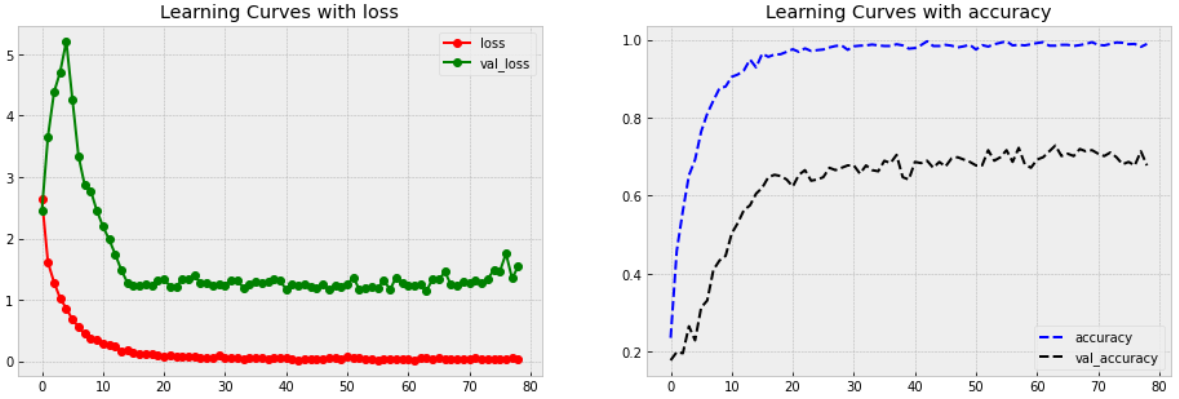


Fig. 2 Learning Curves with loss and accuracy for CNN model

1. *Transfer Learning*

The third model applies transfer learning. One pre-trained model we used was Xception in a Keras application. Xception, proposed by François Chollet in 2016, has a special type of layer called depthwise separable convolutional layers. It is a powerful model pre-trained by image net.

We first set the weights to those learned on ImageNet, specifying the input shape of the RGB image as (300, 300, 3). We do not use the top layer because at the end we will add an output layer with 10 neurons identical to the previous two models to suit our flower classification task. In addition to this, we also freeze the weights of the pre-trained Xception model. Freezing layers means that their weights are not updated during training, only the weights of new layers added later will be trained. The GlobalAveragePooling2D layer pools spatial information by taking the average of each feature map.

Since this is a pre-trained model, we need to adjust the learning rate when compiling to make it more suitable for our task. By observing the learning curves, we can find that when using a high learning rate of 0.01, the model uses fewer epochs to achieve low loss, but the validation does not converge well. Using a low learning rate of 0.0001 will cause the model to reduce training loss and validation loss at a slow rate, making the accuracy improvement very slowly. And after it reaches the upper limit of 100 epochs, it even performs worse than the model using a learning rate of 0.001.

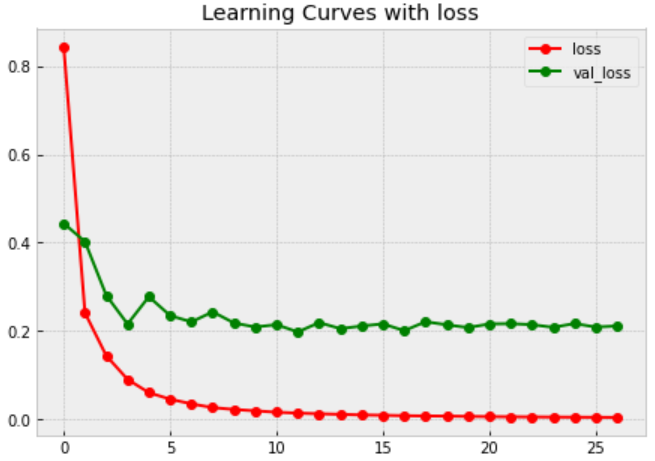
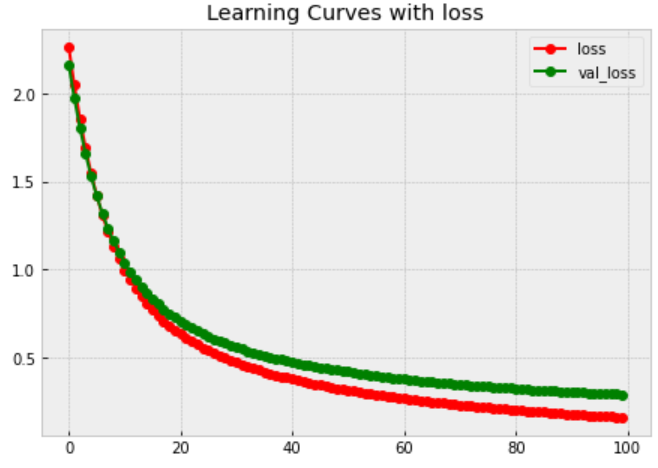
 

Fig. 3 Learning Curves with loss for transfer learning model with learning rate 0.01 (left) and 0.0001 (right).

By selecting the model with a learning rate of 0.001, we observe its performance metrics. This model also achieved 1.0 on the training set and approximately 0.93 on the validation set. Compared with ANN with 2 hidden layers and CNN, the accuracy of the model on the validation set is much greater than them.

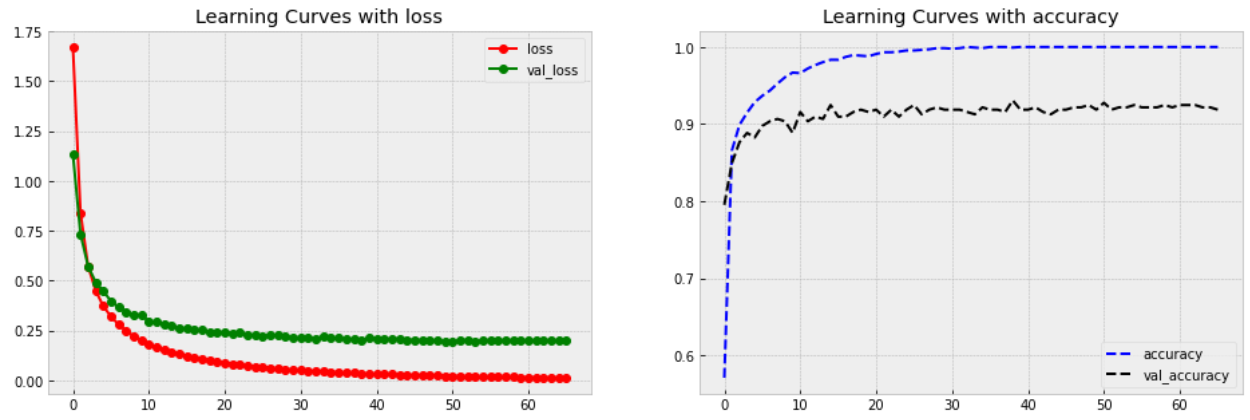


Fig. 4 Learning Curves with loss and accuracy for Transfer Learning model

1. *Comparison of 3 models*

By comparing the performance indicators of the three models on the test set, we can know which model is most suitable for the flower classification task. ANN is ANN with 2 hidden layers; CNN is custom CNN; TL is Transfer Learning model with Xception.

1. Three models’ Performance

|  |  |  |
| --- | --- | --- |
|  | Validation Accuracy | Test Accuracy |
| ANN | 0.58 | 0.54 |
| CNN | 0.65 | 0.68 |
| TL | 0.93 | 0.94 |

We already know that all three models can reach 1.0 in the training set, but the ANN and CNN models have severe overfitting. By comparing the accuracy of each model on the validation set and the accuracy on the test set, we found that transfer learning with Xception has the highest accuracy and is far ahead of the ANN and CNN models. This means that transfer learning with Xception is the optimal model for the flower classification task.

## Car Detection

For this task, our goal is to train a car detection artificial neural network using training samples and make predictions on images under test. This problem applies transfer learning of the pre-trained model VGG16.

The VGG16 model pre-trained on the ImageNet dataset is used for transfer learning. VGG16 is a convolutional neural network for classification and detection. Because the VGG16 architecture is very simple and classic, it has powerful detection capabilities, and it is an excellent model that was born in recent years and is widely used. First, we only consider the case where at least one car is present in the image.

To preprocess the data, we first normalize the pixel values of the input images in the training set to scale them to the range [0, 1]. The target values in the training set are then normalized. Next, we split the training data into 90%/10% training data and validation data. By allocating more training data, we hope that the model can gain a wider range of capabilities to correctly identify car locations.

The ‘input\_tensor’ of the VGG16 model is set to accept input images of shape (224, 224, 3), which are common sizes for images in computer vision tasks. VGG16 is then loaded using the pre-trained ImageNet weights, which do not include the final fully connected classification layer header. And we need to freeze all layers in the pre-trained model and make them untrainable to ensure that we do not need to retrain the complete VGG16 model. I then perform network surgery by building a new fully connected layer header that will output four values corresponding to the upper left and lower right bounding box coordinates of the object in the image.

This time we apply batch normalization and resize to match the expected input size of the pretrained VGG16 model. Flatten the output of the pretrained VGG16 model and add several dense layers with SELU activation and dropout for regularization. The selection of activation function and weight comes from the hyperparameters mentioned in the flower classification task that are advantageous for images. The final output layer has 4 units representing 4 bounding box coordinates and uses sigmoid activation since this is a binary classification task.

Next step is compile and fit the prepared VGG16 model. Also using the Nadam optimizer, the learning rate is initially set to 0.001 based on empirical success. To suit this task we set the loss function to mean squared error. Also apply early stopping and auto save best model according to validation loss. Training was set to run for up to 100 epochs with a batch size of 16 samples per batch. This is because a model trained with smaller batches may generalize better to unseen data. Smaller batch sizes introduce additional noise during training, which can act as a form of regularization.

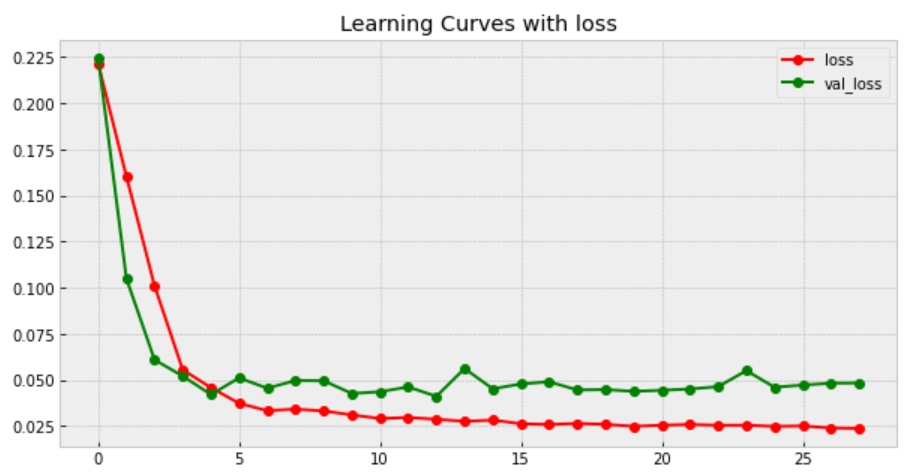


Fig. 5 Learning Curves with loss for VGG16 model on training set

After training, performance on the test set is reported in the form of quantitative and qualitative measures. For qualitative measurements, we will select ten images containing cars in the test set, set our own test labels via MakeSenseAI, and then visualize the results. For quantitative metrics, overlapping regions of interest (ROI) are applied. ROI is calculated as the overlap area between the test bounding box and the predicted bounding box divided by the area of the test bounding box. Accuracy scores range from 0 to 1. We loaded the model into a test file and selected 10 images with labels we created in the test dataset. These labels were created by MakeSenseAI. The accuracy score of ROI in ten images is [0.63, 0, 0.41, 0.40, 0.60, 0.55, 0.35, 0.86, 0.47, 0.83]. The average accuracy is 0.51.

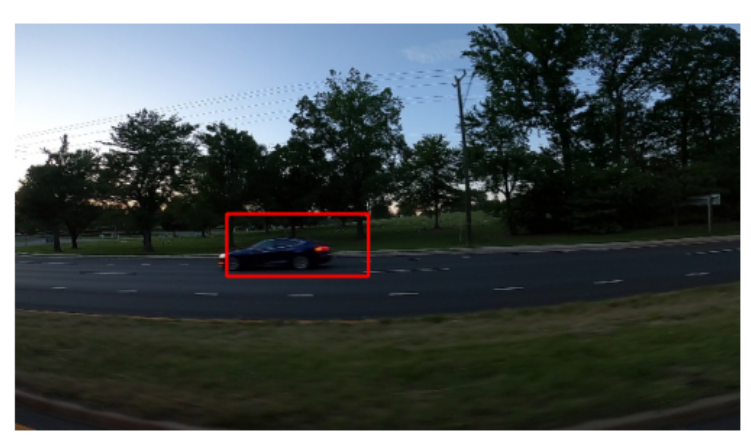
 

Fig. 6 Well performance case from test set

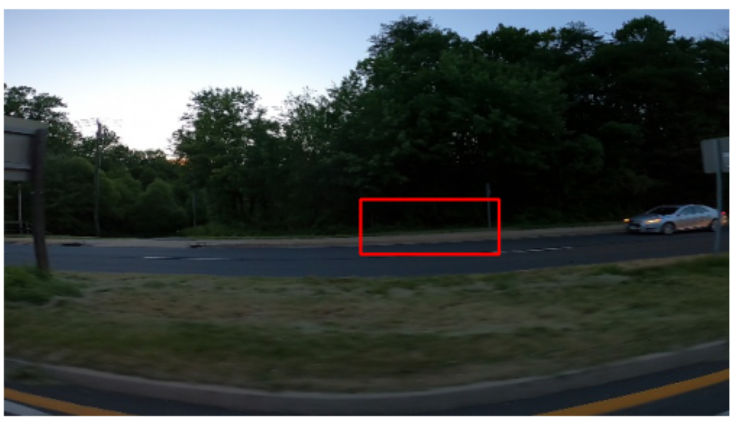
 

Fig. 7 Bad performance case from test set

Through visualization, we found that when the vehicle is located very close to the left or right, the model cannot identify the vehicle well. By comparing the bounding box and ROI accuracy, we can also find that the bounding box cannot contain the entire vehicle well. This results in that even if the bounding box contains part of the vehicle, it cannot obtain high ROI accuracy. I also selected some images that contained multiple cars, and the bounding boxes performed very poorly. It tells us that using only a CNN model (VGG16 is a CNN model) will not perform well in this car detection task.

In scenarios where images have no cars, we have two approaches for training models. In the first approach, one model is dedicated to image classification, assessing whether a car is present. Upon a positive determination, a second model is activated to handle the specific task of car detection. Alternatively, in the second method, all images are trained collectively, and specific bounding box coordinates are assigned for images where no cars are present. For instance, a common practice is to set the label [0,0,0,0] for these particular images.

# conclusion

Through experiments we found that the most suitable model for the flower classification task is transfer learning that uses Xception. It has a far higher accuracy than ANN with 2 hidden layers and custom CNN. For the car detection task, even if we use transfer learning with VGG16, it still only has about 50% ROI accuracy. This task requires a more demanding neural networks model to accurately identify cars, which requires a lot of experiments and a better architecture to achieve