Camouflaged Object Detection

Introduction

Our established goal was the detection of camouflaged/non-camouflaged objects. An object in this context refers to an animal(s) or human(s). We have a collection of images from our dataset that are the given images that our models must classify into one of the three classification: camouflaged object, non-camouflaged object and absent object. We use three different architectures to attempt success with our previously defined goal.

Github

https://github.com/marquezjorge/456-Camo.git

Dataset

For our project we used a dataset that was amalgamtation of 3 different datasets: CAMO,MS-COCO,and COD10K. Each dataset gave of a collection of images for one of the class. The CAMO dataset contained 1250 images of camouflaged objects. 1250 images of non-camouflage objects were collected from the MS-COCO dataset. A final 1250 images were collected from the COD10K dataset to make up the images in the background class. We found it necessary to combine these datasets to make these 3 classes because any practical application of a camoflauged object detector should be able to comptenetly perform 2 basic requirements: Identify that the exists an object in the image and identify whether the object is actively attempting to conseal itself or not.

Related Work

AlexNet

The architecture of AlexNet consists of five convolutional layers and three fully-connected layers. Each convolutional layer is composed of several convolutional kernels with a size of 11x11, 5x5 and 3x3 respectively. The activation function applied to each convolutional layer is the Rectified Linear Unit (ReLU) which is a nonlinear function used to introduce non-linearity into the network. The fully-connected layers have a size of 4096, 4096 and 1000 neurons respectively. The output layer of the network is a softmax layer which is used to classify the input image into one of the 1000 classes.

AlexNet also uses dropout to reduce overfitting. Dropout randomly sets a fraction of output units to zero during training, which reduces the number of parameters and prevents the co-adaptation of neurons. Overall, AlexNet is an influential architecture that has revolutionized the field of deep learning for computer vision and has helped to set the path for modern deep learning techniques.

VGGNet

VGGNet is a convolutional neural network (CNN) architecture developed by the Visual Geometry Group (VGG) at Oxford University, and is widely used for image classification and recognition. It was the runner-up in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2014, and since then has become a popular choice for many tasks.

The VGGNet architecture consists of 16 to 19 layers of 3x3 convolutional kernels followed by 2x2 max pooling layers, interspersed with ReLU activation functions. The model also includes 3 fully-connected layers at the end, with a final softmax classification layer. All convolutional layers use a stride of 1, and all max pooling layers use a stride of 2.

The main feature of the VGGNet architecture is its simplicity. The network does not have any particularly novel or complex components, and it does not rely on any special initialization or training techniques.

LeNet

LeNet is a convolutional neural network (CNN) that was developed by Yann LeCun as a seminal work in the field of deep learning. First published in 1998, it is widely credited as one of the earliest successful applications of CNNs. LeNet is commonly used in the field of computer vision and is best known for its use in handwritten digit recognition.

LeNet is composed of several convolutional and pooling layers, followed by fully-connected layers which are used to classify the input data. It consists of 7 layers in total, including the input layer. In the convolutional layers, the input image is scaled down by applying a filter to the input. This filter is called a convolutional kernel and is used to identify features such as edges and corners. The output of this layer is then passed to a pooling layer. Pooling layers allow the network to reduce the size of the output without losing important information.

The next layer is a fully-connected layer which is used to classify the input data. This layer is composed of neurons which are connected to all of the neurons in the previous layer. Finally, the output layer is used to generate the classification of the input data.

Methodology

Creating the Model: LeNet Convolutional Neural Network

This model is a 9-layered convolutional neural network using Keras. The model follows the structure of a simple Lenet model. It takes in a 32 x 32 x 3 input. The input is passed through 3 convolutional layers each with Relu activation. The first two convolutional 2d layers are each

followed by a max pooling and dropout layer. The image is then flattened and passed through 2 fully connected layers with reLu activation. Finally the output is passed through the output layer with a softmax activation.

Creating the Model: Alexnet Convolutional Neural Network

Following the architecture specified in Alex Krizhevsky's ImageNet Classification with Deep Convolutional Neural Networks a model was developed. It takes 224x224x3 image inputs and is composed of 8 layers in the following order: CONV1, MAX POOL1, NORM1, CONV2, MAX POOL2, NORM2, CONV3, CONV4, CONV5, MAX POOL3, FC6, FC7, FC8. For this particular implementation an input size of 227x227x3 was arbitrarily chosen, a final fully connected layer with an output of 3 classes was added.

Creating the Model: VGGNet Convolutional Neural Network

This model is built using the results of the research article "Very Deep Convolutional Networks for Large-Scale Image Recognition." The findings showed that increasing depth to 16-19 weight layers in a convolutional neural network model with (3x3) filters can lead to significant improvements. It takes an input of size 224x224x3. VGG16 has 16 layers, 13 convolutional and 3 fully connected layers. All the convolutional layers have filters of size 3x3. The first 2 convolutional layers have a feature map size of 64. These are followed by a max pooling layer with stride 2x2. The next 2 layers have a feature map size of 128, and are followed by a max pooling layer with stride 2x2. Layers 5, 6, and 7 have a feature map size of 256, and are followed by a max pooling layer with stride 2x2. Layers 8, 9, and 10 have a feature map size of 512, and are again followed by a max pooling layer with stride 2x2. The last 3 convolutional layers are identical to the previous and followed by a max pooling layer with stride 2x2. Layers

14 and 15 are fully connected layers of 4096 units. Layer 16 is a softmax output layer of 3 units for the 3 categories.

Results & Analysis

LeNet Results & Analysis

LeNet model was originally designed for a dataset of grayscale images of letters. Due to this there was an inherent expectation that this model would have a challenge performing well on a dataset that is so different from what it was designed for. The model did follow our expectations of performing the worst of the 3 models but the performance gap was not as great as originally hypothesized. The model achieved an accuracy of 72.11% and a loss of 0.75195. As shown in Figure 1 and 2., ss the model was being trained, the accuracy and loss for the training set followed a somewhat linear incline/decline. The validation set however followed a relatively linear path and then hit a threshold where it began to plateau. The accuracy plateaued around 65-70% while the loss plateau at about 70%. Once the model hit those numbers increasing the number of epochs did not make any noticeable difference to the accuracy or loss at all. Due to this the number of epochs was limited to 10.

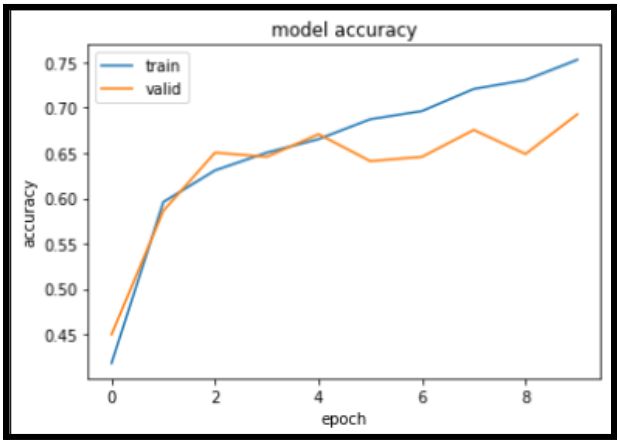

Figure 1.

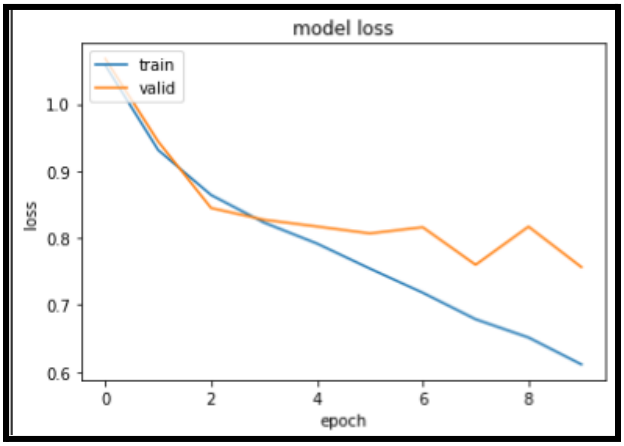

Figure 2.

AlexNet Results & Analysis

Epochs: 30
Learning Rate: 0.001
Optimizer: Stochastic Gradient Descent (SGD)
Loss Function: Sparse Categorical Cross Entropy

```
Epoch 45/50
71/71 [==============================] - 77s 1s/step - loss: 0.4240 - accuracy: 0.8258 - val_loss: 0.9285 - val_accuracy: 0.6160
Epoch 46/50
71/71 [==============================] - 77s 1s/step - loss: 0.3989 - accuracy: 0.8418 - val_loss: 1.0305 - val_accuracy: 0.7107
Epoch 47/50
71/71 [==============================] - 77s 1s/step - loss: 0.3832 - accuracy: 0.8480 - val_loss: 1.1014 - val_accuracy: 0.5907
Epoch 48/50
71/71 [==============================] - 77s 1s/step - loss: 0.3334 - accuracy: 0.8809 - val_loss: 0.8675 - val_accuracy: 0.7000
Epoch 49/50
71/71 [==============================] - 77s 1s/step - loss: 0.3467 - accuracy: 0.8662 - val_loss: 0.9624 - val_accuracy: 0.6747
Epoch 50/50
71/71 [==============================] - 77s 1s/step - loss: 0.3519 - accuracy: 0.8556 - val_loss: 1.0907 - val_accuracy: 0.5947

Process finished with exit code 0
```

Original AlexNet implementation details call for a batch size of 128, an initial learning rate of 0.01, and 90 epochs. For this particular implementation all of those defaults were changed, partly due to hardware limitations and the size of our dataset. It was observed that increasing or decreasing the number of epochs beyond or less than 30 did not positively affect training, the same applied to learning rate. In regard to the optimizer, it was a better choice to use SGD as opposed to the keras default, Adam. Training the model with the Adam optimizer yielded lower validation accuracy, the correct optimizer choice, SGD, was self-evident through trial.

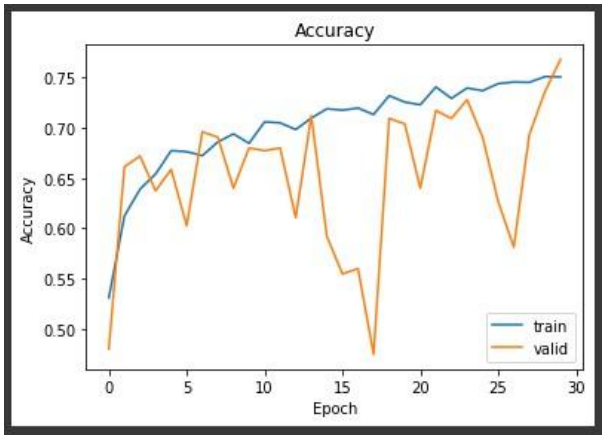

Figure 3.

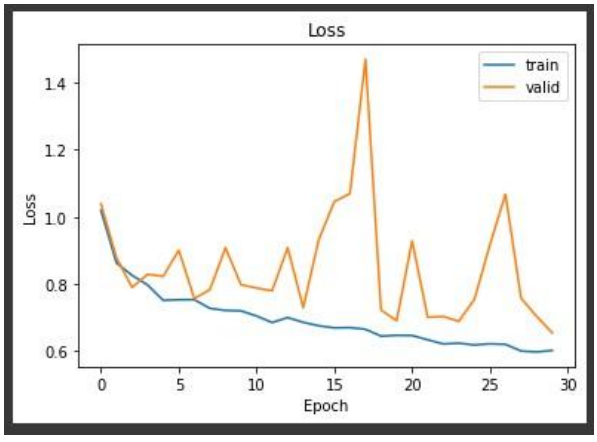

Figure 4.

Accuracy for AlexNet is displayed in Figure 3. As is evident, there were spikes in the accuracy at certain intervals, this implies that the model was experiencing severe overfitting only at certain points. Figure 4 shows the loss for the model, note the large spike in loss around epoch 15 and its corresponding decrease in accuracy shown in Figure 3. This could mean that some of the data samples were of particularly bad quality, it may be possible to improve results if those data samples can be identified and removed from the dataset. The final validation accuracy was 0.7680 and validation loss was 0.6483.

VGGNet Results & Analysis

Epochs: 60
Loss: Categorical Crossentropy
Optimizer: Stochastic Gradient Descent with learning_rate=0.001
Batch size: 32

```
Epoch 56/60
94/94 [==============================] - 25s 258ms/step - loss: 0.6201 - categorical_accuracy: 0.7513 - val_loss: 0.6948 - val_categorical_accuracy: 0.7200
Epoch 57/60
94/94 [==============================] - 25s 258ms/step - loss: 0.6266 - categorical_accuracy: 0.7417 - val_loss: 0.6838 - val_categorical_accuracy: 0.7333
Epoch 58/60
94/94 [==============================] - 25s 261ms/step - loss: 0.6107 - categorical_accuracy: 0.7490 - val_loss: 0.7014 - val_categorical_accuracy: 0.7200
Epoch 59/60
94/94 [==============================] - 25s 261ms/step - loss: 0.6203 - categorical_accuracy: 0.7397 - val_loss: 0.7157 - val_categorical_accuracy: 0.7387
Epoch 60/60
94/94 [==============================] - 25s 262ms/step - loss: 0.6115 - categorical_accuracy: 0.7463 - val_loss: 0.6559 - val_categorical_accuracy: 0.7547
```

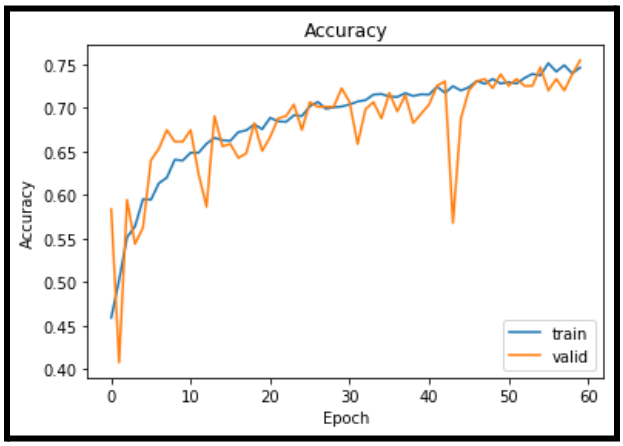

Figure 3.

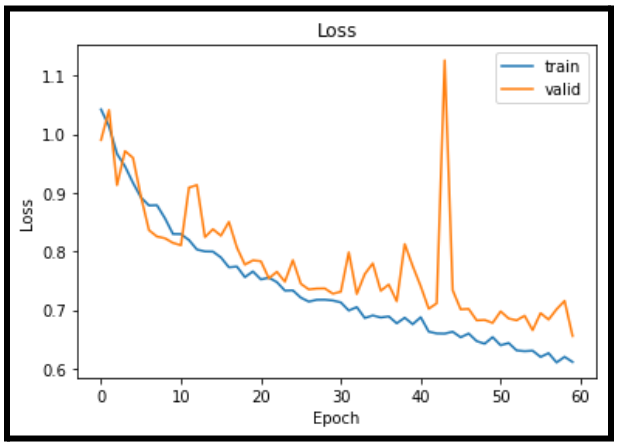

Figure 4.

The architecture designed in the research article “Very Deep Convolutional Networks for Large-Scale Image Recognition” uses a batch size of 256, but it was observed that the batch size did not affect accuracy that much and instead used a batch size of 32. After the accuracy reached

70% its rate of growth per epoch slowed down a lot. The accuracy could be improved with more powerful hardware and time as 60 epochs used a lot of GPU power and took a significant amount of time. At first 0.01 was used for the learning rate, but it was observed that setting it to 0.001 resulted in significantly improved accuracy scores. Using momentum with a value of 0.9 as they did in the article resulted in slightly worse accuracy so it was not used. The results show overfitting, and data augmentation was used including random flipping and rotating of the images to try to account for it.

Discussion

Initially, we expected VGGNet to have the best metrics and LeNet was expected to yield the worst results. In practice the highest accuracy was achieved with AlexNet, followed by VGGNet and lastly LeNet. However our original hypontize does manifes in the ranking of loss wih VGGNet having the lowest.

Model	**Accuracy**	**Loss**
LeNet	0.7211	0.75195
AlexNet	0.7680	0.6483
VGGNet	0.7307	0.6115

Conclusion

The dataset was a collection of 3750 images split between 3 classes: camouflaged object, non-camouflaged object, and background (no object). A successful model nust not only be able to distinguish whether an object was camouflage or not but also whether the images contain an object in the first place. We created 3 models based on 3 different architectures: LeNet, AlexNet,

and VGGNet. Each of these slightly resembles each other but the following model makes significant improvement over the previous. Due to this knowledge, the hypothesized ranking of performance would follow this order as well, which is determined to be an accurate conclusion.

Work Assignments

Melissa	Model: Implemented and trained LeNet model Report: Wrote Dataset, LeNet Convolutional Neural Network , LeNet Results & Analysis, Conclusion sections
Jorge	Implemented and trained AlexNet model Report: Wrote Related Work, AlexNet Convolutional Neural Network , AlexNet Results & Analysis sections
Zach	Implemented and trained VGGNet model Report: Wrote Introduction, VGGNet Convolutional Neural Network, VGG Net Results & Analysis, Discussion sections