Master in Computer Science

Technique of AI

Rapport

Classification of images

Auteurs

AYADI Mustapha

ZHU Yi

IZMAR Soumaya

Professeurs

BERSINI Hugues

2022-2023

Contents

Introduction

In our work, we try to improve the "Cat or Dog" project aiming at classifying images of cats and dogs. And after that, we also try to apply the same technique to classify dogs by their races.

In the project, we use a Convolutional Neural Network (CNN), which is a type of neural network commonly used in image and video recognition tasks. It is inspired by the organization of the visual cortex in animals, where neurons are arranged in layers that respond to visual stimuli in different ways.

In a CNN, the input image is fed into a series of layers, each of which applies a set of filters (also called kernels) to the input. These filters perform convolutions on the input, which means they scan the image for specific patterns or features.

As the input passes through each layer, the filters become increasingly complex, detecting more and more abstract features. The output of each layer is a set of feature maps, which represent the locations of the detected features.

The final layers of the CNN typically include one or more fully connected layers, which use the features detected by the previous layers to classify the image. The output of the final layer is a vector of probabilities, indicating the likelihood that the input image belongs to each of the possible classes.

Transfer Learning

Transfer learning is a technique in machine learning that involves taking a pre-trained model, and using it as a starting point for a new model, rather than training a model from scratch. Transfer learning can be particularly useful in situations where the amount of labeled training data is limited, or when training a new model from scratch would be prohibitively time-consuming or resource-intensive.

There are some common steps we followed:

- Select a pre-trained model (We choose AlexNet, Resnet)
- Choose the layers to transfer
- Re-train the transferred layers
- Evaluate and refine

As we have enough data for training, we decide to train models from scratch and also by using transfer learning, after that, we have done a comparison of the performance.

Dataset and Famous (pre-trained) model

Datasets

The dataset used to train the model to classify cats and dogs is composed of 10 000 coloured images of cats and dogs [1]. Figure 1 shows an example of images in the dataset.

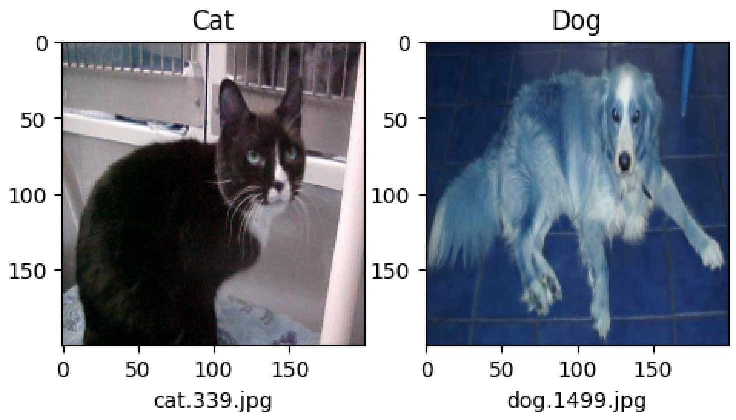


Figure 1: Example of images of cats dogs dataset

The other dataset used is a dataset composed of dogs breed provided by FastAI [2] containing 13000 coloured images. It is already sliced in train/validation set, and each dog breed has a unique ID instead of their breed name. Figure 2 shows an example of images in the dataset.

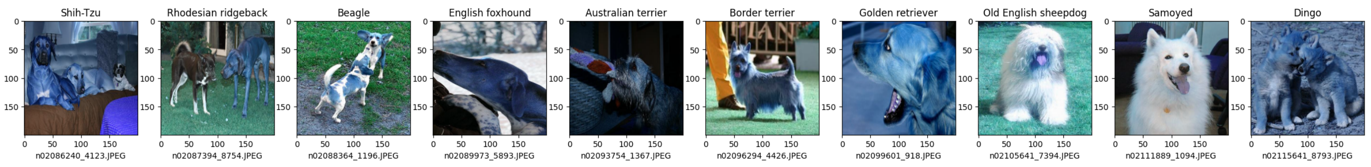
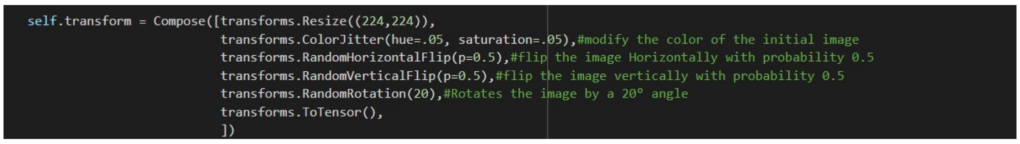


Figure 2: Example of images of dogs breed dataset

Data augmentation

Data augmentation consists of increasing the size of the training set using the data in the training set. In the case of datasets containing images, this manipulation consists of making minor modifications to the base image in order to obtain a new one. This technique also reduces overfitting and increases the size of the dataset. The torchvision.transforms module offers functions for data augmentation. Figure 3 shows the code used during the project.



```
self.transform = Compose([transforms.Resize((224,224)),
                          transforms.ColorJitter(hue=.05, saturation=.05),#modify the color of the initial image
                          transforms.RandomHorizontalFlip(p=0.5),#flip the image Horizontally with probability 0.5
                          transforms.RandomVerticalFlip(p=0.5),#flip the image vertically with probability 0.5
                          transforms.RandomRotation(20),#Rotates the image by a 20° angle
                          transforms.ToTensor(),
                          ])
```

Figure 3: Example of images of cats dogs dataset

Alexnet

Alexnet [3] is a convolutional neural network (CNN) designed by Alex Krizhevsky, hence its name. The success of this architecture comes from the fact that it won the Large Scale Visual Recognition Challenge (LSVRC) in 2012. The challenge is to have the best accuracy on several image recognition challenges. This architecture is more than 10 years old but is still a competitive architecture in image recognition. Alexnet is an architecture that contains **8 layers**; **5 convolutional layers** and **3 fully connected layers**, figure 4 summarises this. At each layer we can find the activation function **ReLu** except for the last one where it is replaced by **Softmax** where its utility is to normalize the output of a sum of points to a probability of belonging to a class (in our case, a probability that the input image is a cat or a dog). We also have the **Max-Pooling** function which is applied after the first, second and fifth convolution layers. Finally, the **Dropout** function is applied to the first two fully connected layers.

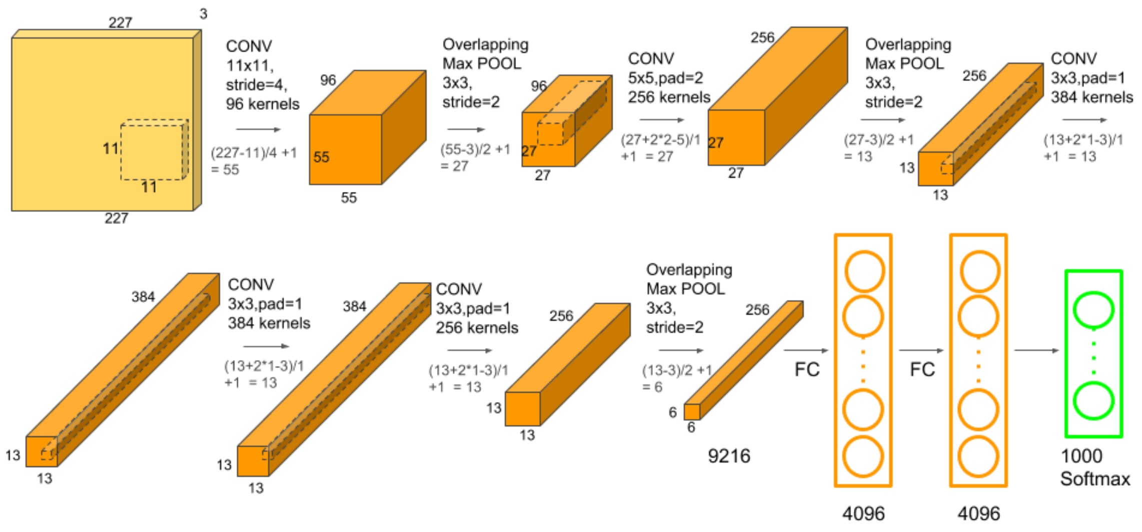


Figure 4: Scheme of AlexNet architecture

One of the weaknesses of AlexNet is that it has over 60 million parameters. It is therefore necessary to do something to avoid overfitting. AlexNet implements two methods to avoid this:

- **Data augmentation**: This technique will be described in more detail later as several models use this technique.

- **Dropout**: This technique "*turns off*" certain neurons with a probability fixed in advance (0.5 in our case). Neurons that are "*turned off*", do not contribute to the forward pass and do not participate in backpropagation. This means that at each iteration, the architecture is a slightly different one.

ResNet

ResNet [4] (aka Residual Networks) is a classic neural network used for many computer vision real-world tasks. This model won ImageNet challenge in 2015. The Resnet architecture can be divided into 6 parts: Input Pre-processing (stage 1), Stage 2, Stage 3, Stage 4, Stage 5, Fully-connected layer (output)

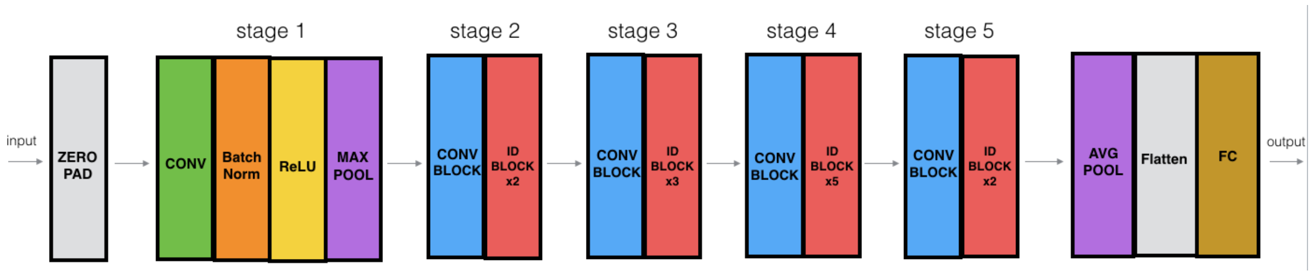


Figure 5: Scheme of ResNet

Every ResNet architecture performs the initial convolution and max-pooling using 7×7 and 3×3 kernel sizes respectively. Afterwards, Stage 1 of the network starts and it has 3 Residual blocks containing 3 layers each. The size of kernels used to perform the convolution operation in all 3 layers of the block of stage 1 are 64, 64 and 128 respectively. As we progress from one stage to another, the channel width is doubled and the size of the input is reduced to half.

For deeper networks like ResNet50, bottleneck design is used. For each residual function F, 3 layers are stacked one over the other. The three layers are 1×1, 3×3, 1×1 convolutions. The 1×1 convolution layers are responsible for reducing and then restoring the dimensions. The 3×3 layer is left as a bottleneck with smaller input/output dimensions.

During the backpropagation stage, the error is calculated and gradient values are determined. The gradients are sent back to hidden layers and the weights are updated accordingly. The process of gradient determination and sending it back to the next hidden layer is continued until the input layer is reached. The gradient becomes smaller and smaller as it reaches the bottom of the network. Therefore, the weights of the initial layers will either update very slowly or remain the same.

The idea used by ResNet to avoid this problem is by using skip connections. They are used to bypass one or more layers in the neural network. The idea behind skip connections is to allow the gradient to flow more easily through the network during training and to address the problem of vanishing gradients that can occur in very deep neural networks. If any layer ends up damaging the performance of the model in a plain network, it gets skipped due to the presence of the skip-connections.

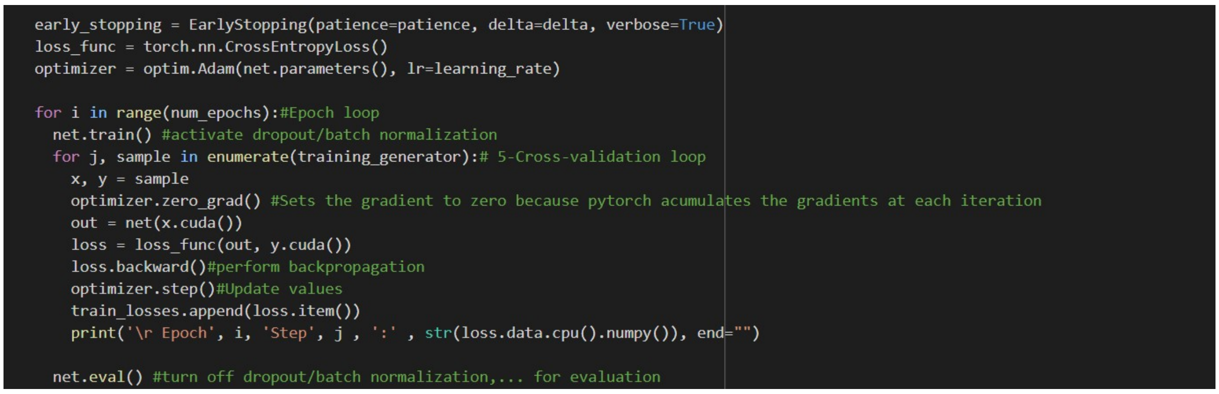
Pretrained model vs self-trained model

In this section, we will compare the pre-trained and self-trained models of Alexnet and Resnet on the cats and dogs dataset. The comparison tool will be the accuracy of both models. We use the *Pytorch* library instead of *Keras*. Both libraries are equivalent to our needs, but we find the Pytorch syntax easier to understand and use. The *Torchvison* package is a package provided by Pytorch. It contains a set of datasets as well as model architectures, and common image transformations.

Training procedure

We have talked about the models and their architecture, but we have not yet discussed the training procedure. Before talking about the procedure itself, we need to define the loss function and the optimizer that we will use. In this project, we want to compare the different models pre-trained on our dataset. So we decided to use the same loss function and the same optimizer. We use the *Cross entropy* as the loss function and *Adam optimizer* for the optimizer. In figure , a commented code snippet shows the training procedure. We train our model on 80% of the data and test with the remaining 20% for "cats and dogs". For "dog breed", the split is 70% for the training and 30% for the test set.

One thing we have added is in the first line in the code of figure , this is the early stopping. This technique allows us to put in a large number of epochs and make sure that if the model does not improve any more, we stop the training. In our case, we look at the validation loss. If the validation loss does not decrease after a certain number of epochs, the training of the model is stopped and the model is saved at the lowest validation loss (checkpoint). We use an implementation found on *Git*.



```
early_stopping = EarlyStopping(patience=patience, delta=delta, verbose=True)
loss_func = torch.nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(), lr=learning_rate)

for i in range(num_epochs):#Epoch loop
  net.train() #activate dropout/batch normalization
  for j, sample in enumerate(training_generator):# 5-Cross-validation loop
    x, y = sample
    optimizer.zero_grad() #Sets the gradient to zero because pytorch acumulates the gradients at each iteration
    out = net(x.cuda())
    loss = loss_func(out, y.cuda())
    loss.backward()#perform backpropagation
    optimizer.step()#Update values
    train_losses.append(loss.item())
    print('\r Epoch', i, 'Step', j , ':' , str(loss.data.cpu().numpy()), end="")

  net.eval() #turn off dropout/batch normalization,... for evaluation
```

AlexNet implementation and result

By using the pre-trained model provided by torchvision, and using the transfer learning method to map with the cats and dogs classes, we obtain $\approx 94\%$ accuracy. While by training the model ourself, only with the cats and dogs dataset, with a learning rate of $1e-4$ and 50 epochs, we get $\approx 86\%$ accuracy. We therefore have a gain of almost 10 percent in accuracy by using a pre-trained model.

```
import torchvision.models as models

alexnet = models.alexnet(pretrained= True)
alexnet.classifier[6] = nn.Linear(in_features=alexnet.classifier[6].
    in_features, out_features=len(label_dict.keys()))#Change output classes
    to cat and dog
alexnet = alexnet.cuda() #Allow model to train on GPU
avg_train_losses, avg_valid_losses = train(alexnet, learning_rate=1e-4,
    num_epochs=50)
```

Listing 1: Alexnet code

Concerning the validation loss, figure 6 and 7 show the evolution of the validation loss versus the number of epochs for AlexNet pre-trained and self-trained respectively. We can see that for the pre-trained model after 8 epochs, the model does not progress anymore and starts to overfit which causes that at the 18th epoch the early stopper stops the training. For the self-trained model, we see the same thing but the model progresses until the 32nd epoch and stops at the 42nd epochs.

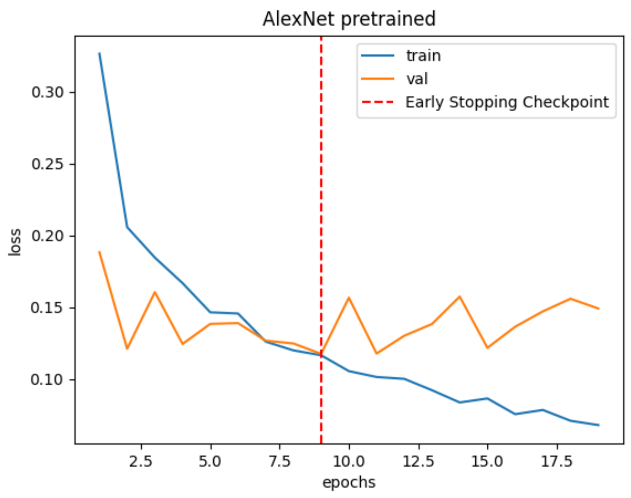


Figure 6: Evolution of the validation loss according to the number of epoch for pre-trained model

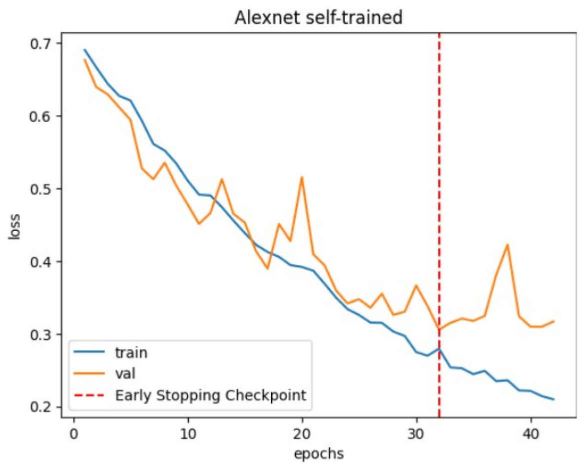
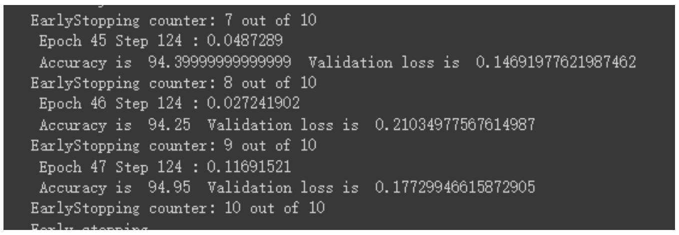
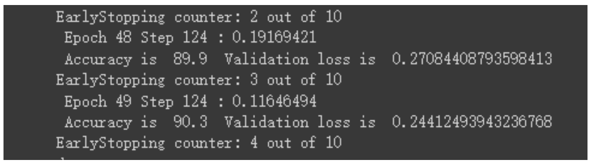


Figure 7: Evolution of the validation loss according to the number of epoch for self-trained model



```
EarlyStopping counter: 7 out of 10
 Epoch 45 Step 124 : 0.0487289
 Accuracy is  94.39999999999999  Validation loss is  0.14691977621987462
EarlyStopping counter: 8 out of 10
 Epoch 46 Step 124 : 0.027241902
 Accuracy is  94.25  Validation loss is  0.21034977567614987
EarlyStopping counter: 9 out of 10
 Epoch 47 Step 124 : 0.11691521
 Accuracy is  94.95  Validation loss is  0.17729946615872905
EarlyStopping counter: 10 out of 10
Early stopping
```

Figure 8: pre-trained ResNet



```
EarlyStopping counter: 2 out of 10
 Epoch 48 Step 124 : 0.19169421
 Accuracy is  89.9  Validation loss is  0.27084408793598413
EarlyStopping counter: 3 out of 10
 Epoch 49 Step 124 : 0.11646494
 Accuracy is  90.3  Validation loss is  0.24412493943236768
EarlyStopping counter: 4 out of 10
```

Figure 9: self-trained ResNet

ResNet implementation and result

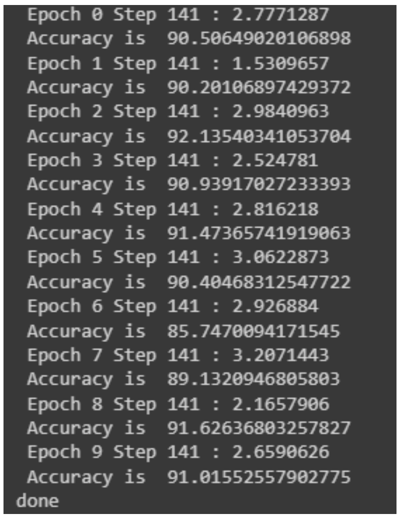
Torchvision pre-trained ResNet50 model is trained with ImageNet [5]. It uses 1.2 million training images, 50,000 validation images and 100,000 test images. Each image in the dataset is labelled with one of the 1,000 categories such as "cat," "dog," "car,", "airplane" etc.

By using the pre-trained model provided by torchvision, and using the transfer learning method with the cats and dogs dataset, we can obtain a 95% accuracy8. While by training the model, only with the cats and dogs dataset, with a learning rate of $1e-3$ and 50 epochs, we can get a 90% accuracy9.

The code snippet for ResNet is similar to the one used for AlexNet.

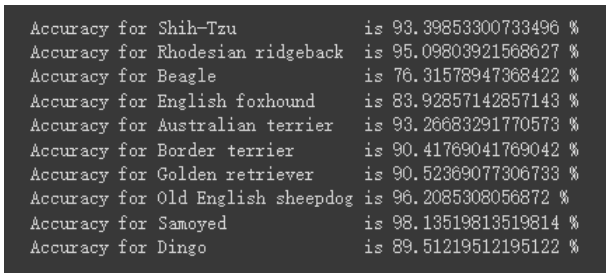
Dog breed

We also try to classify dogs by their breed. The dataset used is mentioned here [2]. To facilitate the training of the model, we used the pre-trained ResNet50 model and changed the last fully connected layer of 1000 outputs to a 10 outputs layer corresponding to the 10 breeds we have.



```
 Epoch 0 Step 141 : 2.7771287
 Accuracy is  90.50649020106898
 Epoch 1 Step 141 : 1.5309657
 Accuracy is  90.20106897429372
 Epoch 2 Step 141 : 2.9840963
 Accuracy is  92.13540341053704
 Epoch 3 Step 141 : 2.524781
 Accuracy is  90.93917027233393
 Epoch 4 Step 141 : 2.816218
 Accuracy is  91.47365741919063
 Epoch 5 Step 141 : 3.0622873
 Accuracy is  90.40468312547722
 Epoch 6 Step 141 : 2.926884
 Accuracy is  85.7470094171545
 Epoch 7 Step 141 : 3.2071443
 Accuracy is  89.1320946805803
 Epoch 8 Step 141 : 2.1657906
 Accuracy is  91.62636803257827
 Epoch 9 Step 141 : 2.6590626
 Accuracy is  91.01552557902775
done
```

Figure 10: global accuracy of the model



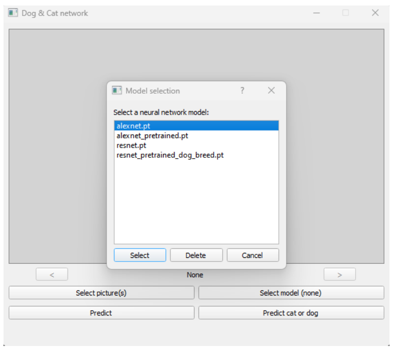
```
Accuracy for Shih-Tzu            is 93.39853300733496 %
Accuracy for Rhodesian ridgeback is 95.09803921568627 %
Accuracy for Beagle              is 76.31578947368422 %
Accuracy for English foxhound    is 83.92857142857143 %
Accuracy for Australian terrier  is 93.26683291770573 %
Accuracy for Border terrier      is 90.4176904176904 2 %
Accuracy for Golden retriever    is 90.52369077306733 %
Accuracy for Old English sheepdog is 96.2085308056872 %
Accuracy for Samoyed             is 98.13519813519814 %
Accuracy for Dingo               is 89.51219512195122 %
```

Figure 11: accuracy per breed

After that, we train the model in the same manner as before, but with a learning rate of 1e-4 and a batch size of 32. As we can see from the figure10, the model can attain a global accuracy of 91%, and in the figure11, we can see that the accuracy for most of the breeds is over 90%, only the Beagle is quite less accurate.

Application

This section is dedicated to the running of the application which is based on the one proposed in the *AI Book*. In figure 12, we can first select the model we want to use from the list proposed. Then, we have to select an image to predict and press the "*Predict*" button to get the prediction with the percentage of confidence of this result. Figure 13 and 14 show an example of prediction for the problem "*cats and dogs*" and "*dog breed*" respectively.



Dog & Cat network

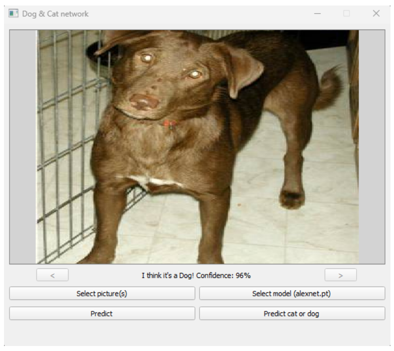
Model selection

Select a neural network model:

alexnet.pt
alexnet_pretrained.pt
resnet.pt
resnet_pretrained_dog_breed.pt

None

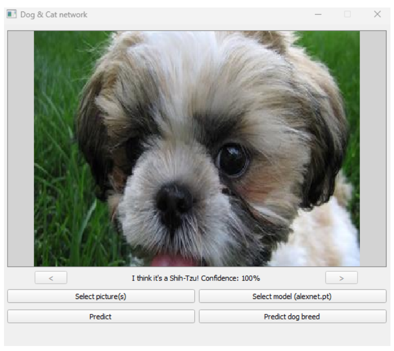
Figure 12:



Dog & Cat network

I think it's a Dog! Confidence: 96%

Figure 13:



Dog & Cat network

I think it's a Shih-Tzu! Confidence: 100%

Figure 14:

Conclusion

Things to improve

We have tried to do model assembling in order to decrease the variance and keep the same performance. Unfortunately we do not have the necessary computational resources. Model assembling requires at least 10 trained models to be interesting to study. We only had *Google Colab* at our disposal and the computation time on GPU is limited to 4h per day. Moreover, the environmental impact was more important to us than running a minimum of 10 models.

We could also implement a *learning rate scheduler*. This technique allows us to adjust the learning rate dynamically. It could allow our model to learn more efficiently, by adjusting the learning rate dynamically. The scheduler helps the network to improve accuracy and reduce losses.

Finally, we could have tested other pre-trained models like *GoogleNet*, *EfficientNet*, *Inception*, and so on. As for the previous point, we were limited by the computational resources at our disposal.

Results

We can see that transfer learning has allowed us to gain accuracy on both models tested. The biggest gain in accuracy is for Alexnet with almost $\approx 10\%$ more accuracy using the pre-trained model compared to $\approx 5\%$ for the pre-trained model from ResNet. If we compare the basic model proposed by the AI Book and our models. We can see that our models have a much higher accuracy than the AI Book even with the self-trained models. Finally, the model we decided to keep is *ResNet pre-trained*. Furthermore, we used ResNet pre-trained for the dog breeds and obtained an accuracy of $\approx 90\%$.

All training Python scripts, datasets and an example application are available in the GitHub reposiroty [6].

Bibliography

[1] CHETANIMRAVAN. dogs and cats images. `https://www.kaggle.com/datasets/chetankv/dogs-cats-images`, 2023.

[2] fast_AI. dogs breed dataset. `https://s3.amazonaws.com/fast-ai-imageclas/imagewoof2-320.tgz`, 2023.

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, may 2017.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[5] standfort vision lab. imagenet. `https://www.image-net.org/`, 2023.

[6] mustivore. Cnn dog(breed) and cat classification. `https://github.com/mustivore/Cat_Dog_Breed`, 2023.