Francisco Javier Sáez Maldonado
franciscojavier.saez@estudiante.uam.es

Image Classification. CNN architectures. Transfer Learning
Deep Learning for Image and Video Processing

2022-04-08

Simple CNN

In this first section, we will present the results obtained by a Simple CNN classifying the digits in the MNIST dataset. Let show some properties about the images in this dataset:

- Train and validation set sizes in MNIST:

	Image height	Image width	Number of Channels	Number of samples
Train	28	28	1	60000
Validation	28	28	1	10000

- Number of trainable parameters of the simple CNN model:

	Number of trainable parameters
Simple CNN	813802

- Show the learning curves for the first 10 epochs. Also indicate the best accuracy obtained and the epoch in which this accuracy is obtained. Comment the conclusions about the evolution of the *loss* in both training and validation sets, with respect to the possible *high-bias* or *overfitting* problems. Would training the model for more epoch improve the performance of the model?

	Highest accuracy (validation)	Epoch with highest accuracy
Simple CNN	99.01	10

As we can see, our model gets a result that is close to 100% accuracy, which means that our model is capable of classifying almost all validation images correctly. Let us see the learning curves:

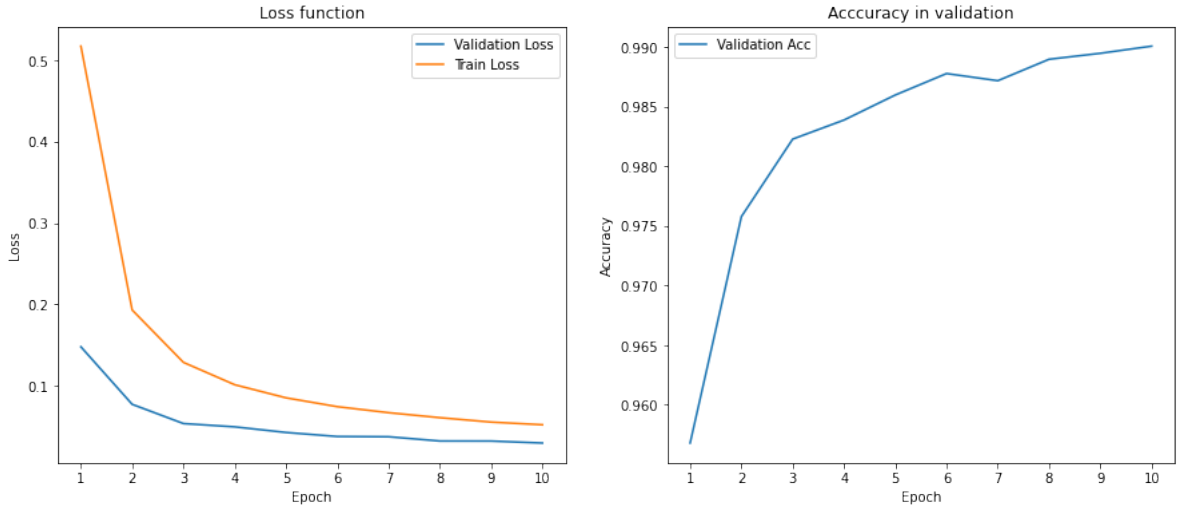


Figure 1: Learning curves for the Simple CNN model.

As we can see, the loss function values are always lower in the validation set. This **makes sense** due to the **dropout**. While training, some neurons are deactivated because of the dropout, leading to *worse* outputs in each batch and, thus, higher loss values. When the validation is performed, **all the weights** are used (dropout is off), so a better middle representation is obtained resulting in a lower loss. In general, we can say that dropout helps us to **avoid overfitting**.

The value of the loss function in both train and validation sets **always decreases**, so we can not say that our model is overfitting yet. Since we are not overfitting yet and the learning curves have not *flattened* (become almost constant), we state that it would be **beneficial** to the model to resume the training for a few more epochs. In fact, we tested this and obtained the following results:

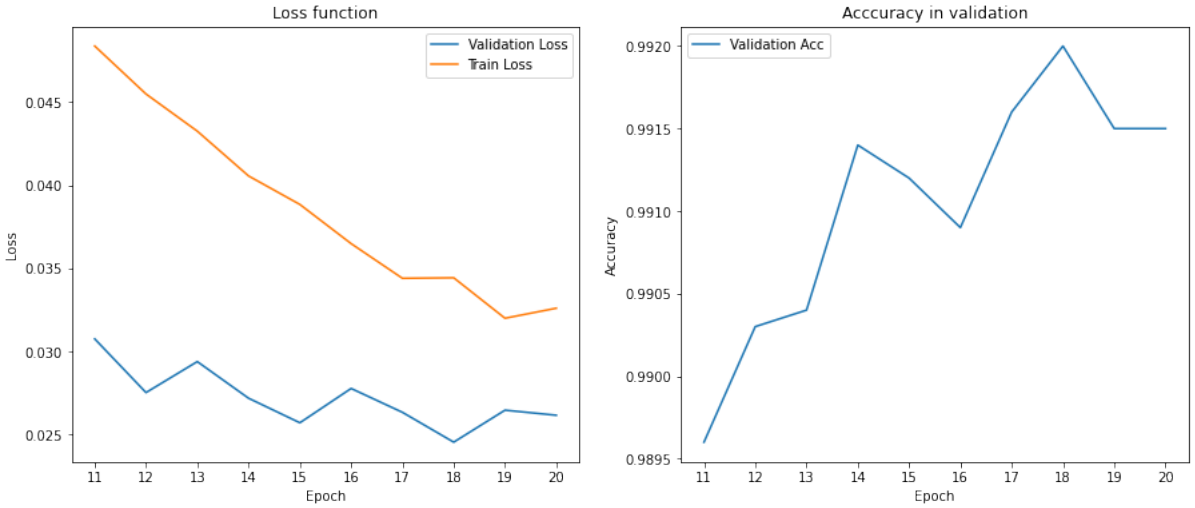


Figure 2: Learning curves for the following 10 epochs.

As we can observe (also looking at the y axis values), the decrease in the loss function and the increase in the accuracy are not relevant. The loss function values in the *train set* even increases sometimes in the last epochs, indicating that the model is possibly already **overfitting**.

We have not observed any **high-bias** problems in any of the cases.

- Show the obtained confusion matrix. Given this confusion matrix, which are the most two confused classes?

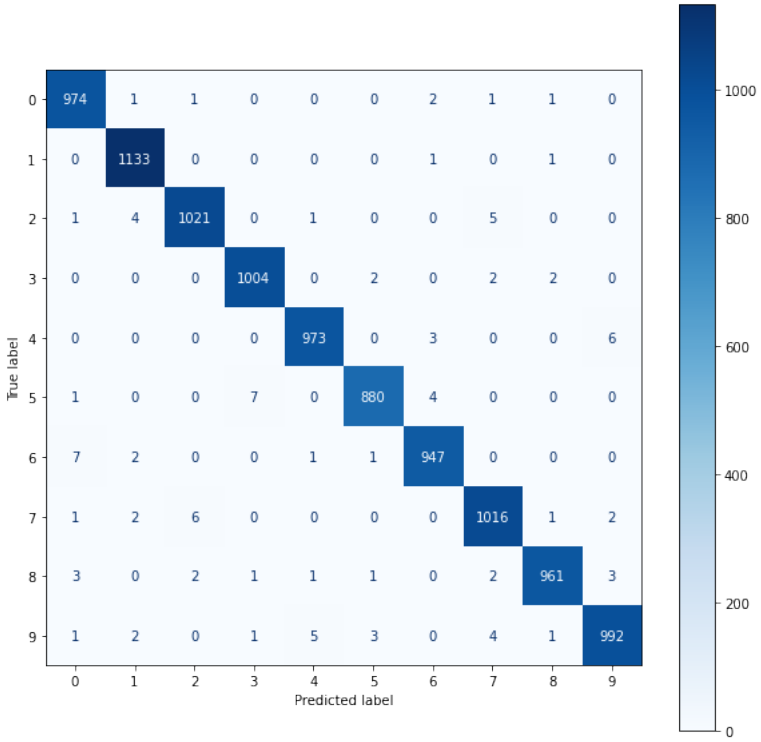


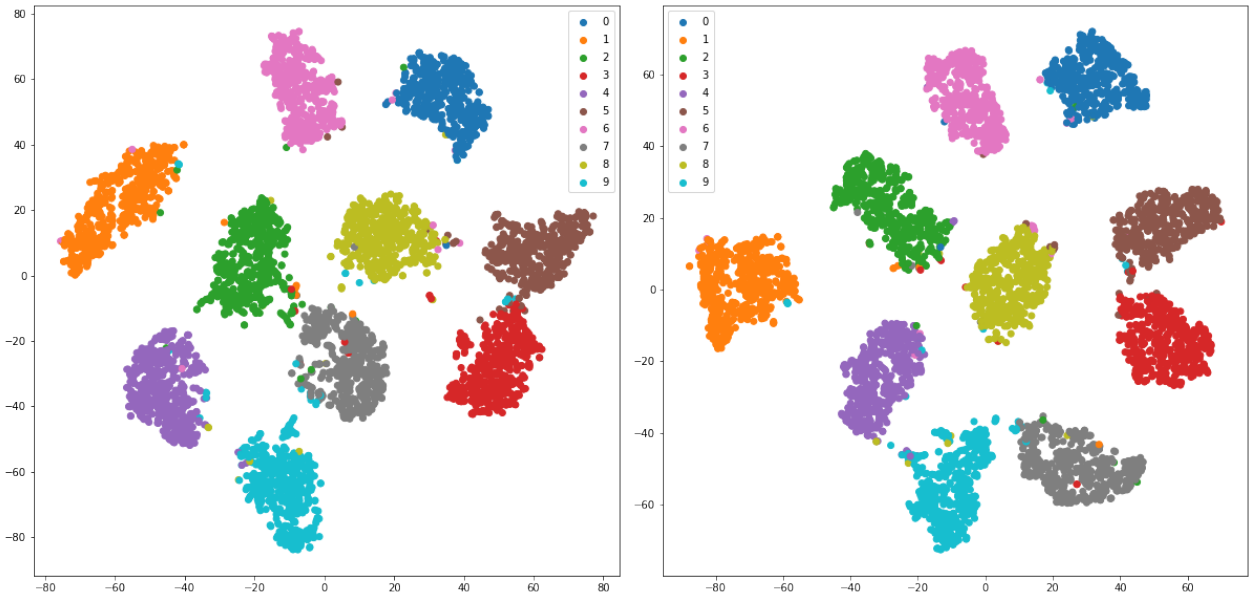
Figure 3: Confusion matrix of the Simple CNN model in MNIST dataset.

As we can see in the confusion matrix, there are two pairs of numbers that are the most confused:

- $6 - 0$. In this case, this confusion is justified due to the rounded shape of both numbers, and if the upper side of the number 6 is not *separated* enough from the rest of the number, the model could be confused.
- $5 - 3$. These two numbers are identical in their lower half and they both have an *"horizontal"* line on the upper half, so the confusion might be justified.

- Comment the differences between the *t-SNE* chart of the representation of the final and intermediate layers of the CNN applied to the validation set. Consider the distance and dispersion between clusters in both representations and its relation with the capability of obtaining a correct classification of the samples.

  Let us firstly present the *t-SNE* representations that we have obtained:



(a) *Intermediate Representation* (b) Output representation

Figure 4: *t-SNE* representations.

  Clearly, the **output** *t-SNE* has more inter-cluster distance and less dispersion in each cluster, so using this representation would provide better results. **However**, we could perfectly use the intermediate representations since we see that the clusters are already **well differentiated** in the intermediate representations. It would be useful to use this intermediate representation if we needed a faster system which also has a good performance.

- Given the differences between the *t-SNE* representation in the different layers, and given the architecture of the CNN, identify the layer of the CNN in which the features are extracted and propose a solution to reduce the Network's complexity with a low penalty in classification accuracy.

  Having a look at the architecture of *SimpleCNN*, we observe that the intermediate features are obtained **after the first linear layer**. Furthermore, the features are obtained in the **convolutional layers** and refined with the first linear layer. As we just saw in Figure 4, the difference between the representations is not huge in terms of clustering separability (a nonlinear model could separate the data quite well since there is a fair amount of distance between the clusters), so **erasing the last linear layer** would be a way to reduce the network's complexity with a low penalty in classification.

  Further study on the intermediate representations obtained by each *convolutional layer* should be done in order to determine if some convolutional layers could be removed.

AlexNet

In this section, we propose a **reduced** implementation of the famous *AlexNet*, and we test it in the dataset **CIFAR10**.

- Include the code used to define the `AlexNet` class. The code used to implement is the following:

```
      class AlexNet(nn.Module):
def __init__(self, output_dim):
    super().__init__()

    self.features = nn.Sequential(
        # First convolutional layer. Use 5x5 kernel instead of 11x11
        # Recall that image input will be 32x32!

        # 5x5 layers
        #-----------
        #in_channels, out_channels, kernel_size, stride, padding
        nn.Conv2d(3, 48, 5, 2, 2),
        nn.MaxPool2d(2), #kernel_size
        nn.ReLU(inplace = True),
        # Subsampling is only performed by 2x2 max pooling layers
        # (not with stride in the  convolutional layers)
        nn.Conv2d(48,128,5,1,2),
        nn.MaxPool2d(2), #kernel_size
        nn.ReLU(inplace = True),

        # 3x3 layers
        #-----------
        # First 3x3 layer
        nn.Conv2d(128,192,3,1,1),
        nn.ReLU(inplace = True),
        # Second 3x3 layer
        nn.Conv2d(192,192,3,1,1),
        nn.ReLU(inplace = True),
        # Third 3x3 layer
        nn.Conv2d(192,128,3,1,1),
        nn.MaxPool2d(2),
        nn.ReLU(inplace = True)
    )

    self.classifier = nn.Sequential(
        # First linear layer
        nn.Dropout(0.5),
        nn.Linear(128 * 2 * 2, 2048), # final conv layer resolution 2x2
        nn.ReLU(inplace = True),
        # Second dropout
        nn.Dropout(0,5),
        # second linear layer
        nn.Linear(2048, 2048),
        nn.ReLU(inplace = True),
        # Last Linear layer. No ReLU
        nn.Linear(2048, output_dim)
    )

def forward(self, x):
    x = self.features(x)
    interm_features = x.view(x.shape[0], -1)
    x = self.classifier(interm_features)
    return x, interm_features
```

In this case, we are using **two sequential blocks** in order to simplify the code of the `forward` method.

- Number of parameters in AlexNet

	Number of trainable parameters
AlexNet	6.199.498

- Learning curves for the first 15 epochs. Also indicate the best accuracy obtained and the epoch in which this accuracy is obtained. Comment the conclusions about the evolution of the *loss* function in both train and test sets. Comment the behavior of this function after epoch 10. Would training the model for more epoch improve the performance of the model?

  Firstly, we see that the highest accuracy in the validation set is the following:

	Highest accuracy (validation)	Epoch with highest accuracy
AlexNet	0.72	14

  We now plot and analyze the learning curves:

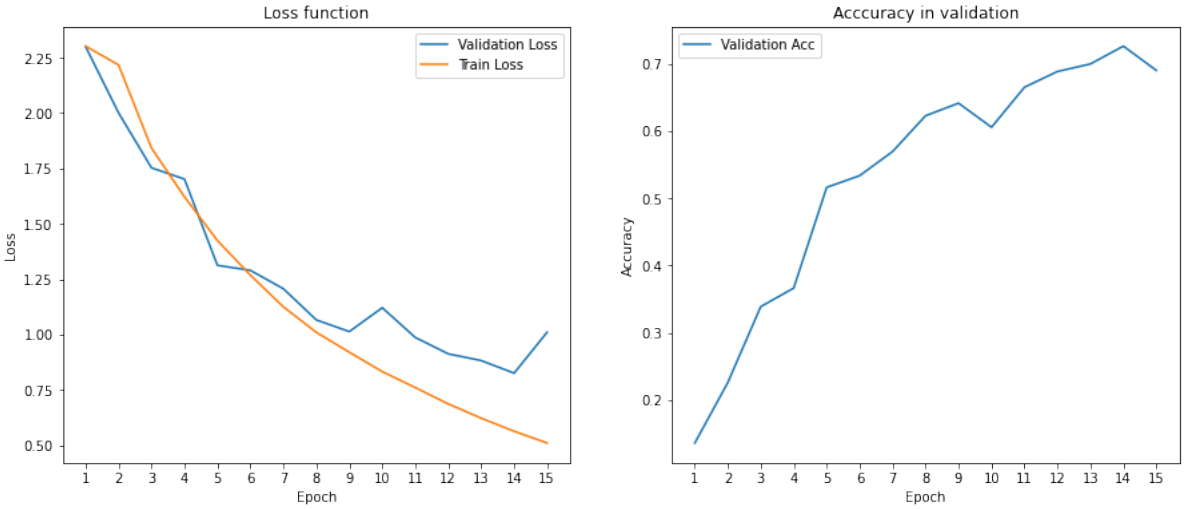


Figure 5: Reduced AlexNet: learning curves for the first 15 epochs.

  As we can see, after epoch 10 we see that the pace of decrease in the loss function in the validation set is much slower than in the previous epochs, which means that our model is already overfitting. Also, as we can see, the increase in the validation accuracy from epoch 10 is much lower than the increase in the previous epochs. With this information, we can state that training the model for more epochs **would not** help our model to obtain better accuracy results, since the model would overfit the train set.

- Show and comment the confusion matrix, taking into account the features of each class images.

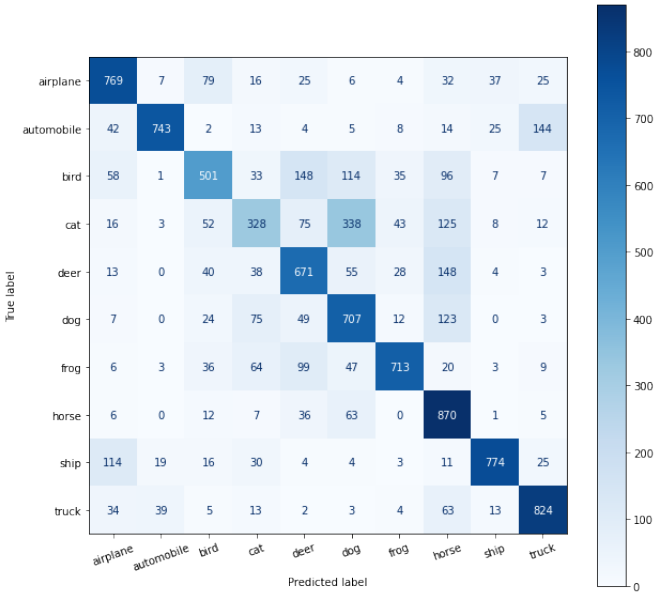


Figure 6: Confusion matrix of Reduced AlexNet in CIFAR10 dataset.

As we can observe, this confusion matrix shows quite worse results than the one presented for the *MNIST* model, which makes sense if we have a look at the accuracies of each of the models.

We can comment the classes that are most confused:

- *Automobile-truck*: The similarity between these two classes is high, since they are both vehicles and have similar features such as rounded shapes (wheels) and defined edges (structure of the vehicle).
- *Cat-Dog*: both animals have approximately same size and similar facial features, although a human could easily distinguish them.
- *Bird-Deer*: this confusion does not make much sense, since the sizes and anatomy of both animals are quite different. The only relation that we could find between the classes is that probably the images of both classes have a *similar background* (surrounded by nature).
- *Horse-Deer*: In this case, the resemblance between the two represented animals is much higher, so the confusion makes sense.

- Show the *t-SNE* representations for the output layer of the network and analyze them taking into account the aspect of the different class images. Compare the results with the representations obtained in the **MNIST**.
dataset.

The representations obtained by the output class are the following:

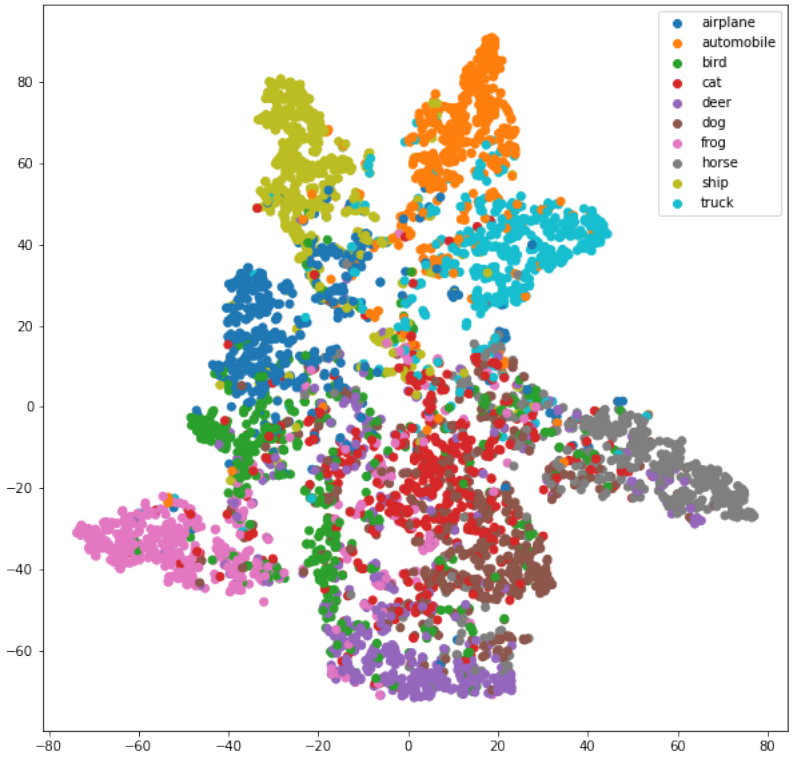


Figure 7: Reduced AlexNet representations by *t-SNE*.

As it can be appreciated after a quick look, the clusters of representations are **much worse** separated than in the previous case. It seems different classes are grouped in *separated* groups, however the amount of noise appearing in each of those groups is much higher than it happened in the *MNIST* case. Furthermore, some of the classes that were being confused have really close *t-SNE* representations (in terms of $\mathbb{R}^2$ distance), as it happens in the case of the classes *Cat-Dog* or *Automobile-Truck*.

In order to improve the robustness of the model in the representation-learning stage, we should probably use data augmentation techniques in order to give the model more examples of the desired classes.

Transfer learning

In this last section, we study the effect of using the pre-trained feature extraction layers of a model in a different problem to see if it provides a good starting point for the new problem. We will use the Alien vs Predator dataset from Kaggle.

- Show the precisions obtained for the different analyzed alternatives.

	Trained from zero	Pretrained + SVM	Finetuning (no data augmentation)	Finetuning (data augmentation)
Accuracy	0.72	0.905	0.935	0.95

Table 1: Obtained results in transfer learning in the Alien dataset.

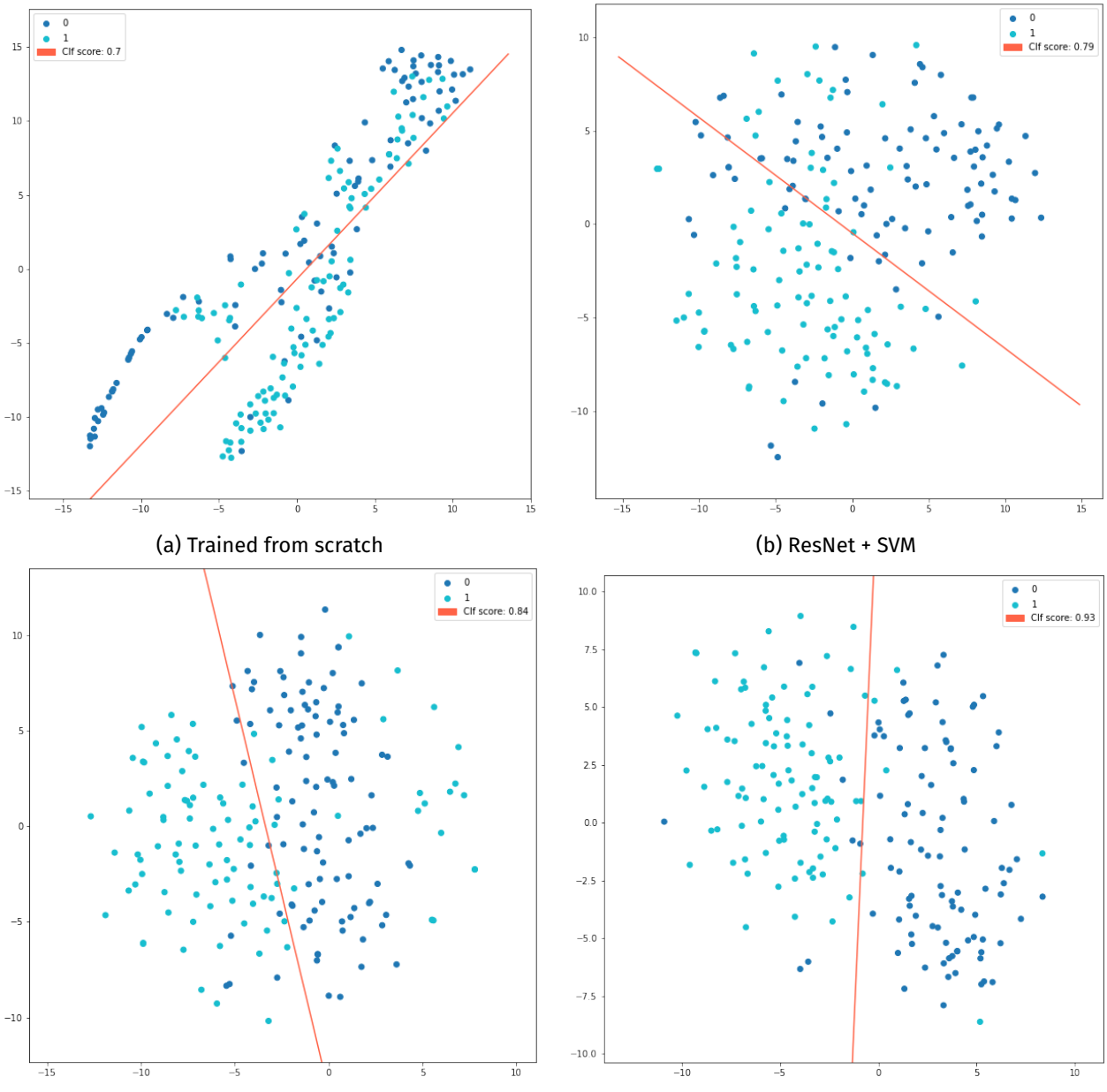
The Table 1 represents the results obtained in each of the stages of the transfer learning process performed. The results show what we expected:

1. Firstly, the trained from scratch model performs worse than using a pretrained model (with no training at all) to extract features and use a classifier on top of that model.
2. Then, it is shown how the performance improves when we *freeze* all layers but the last one and train that layer for a few epochs (which is known as finetuning).
3. Lastly, we check the positive effect of applying data augmentation when training both the last linear layer and the classifier, obtaining the highest accuracy.

All this results were expected, since it is known that ResNet (network used as feature extractor) is one of the most powerful CNN architectures and finetuning the last layers adapts the pretrained weights to improve the performance on a new problem.

- Compare the *t-SNE* representations of the different alternatives. From these representations, comment the difference in terms of linear separability and the accuracy of this linear separability.

We plot the *t-SNE* representations, using a **linear model** to try to separate the classes.



(a) Trained from scratch

(b) ResNet + SVM

(c) Finetune

(d) Finetune + Data Augmentation

Figure 8: *t-SNE* representation of the Alien vs Predator dataset. Linear classifier in each of them.

The accuracy scores obtained by each of the linear models are presented in Table 2.

	Trained from zero	Pretrained + SVM	Finetuning (no data augmentation)	Finetuning (data augmentation)
Accuracy	0.7	0.79	0.84	0.93

Table 2: Obtained results linear classification of *t-SNE* representation in Alien vs Predator dataset.

The analysis using the images or the tables is quite similar. As we can observe, the representations obtained using the trained from scratch model are quite bad (Figure 8a). It is linearly separated in the

bottom part of the image, but in general the points of the different classes are mixed. When we use a pretrained model with a SVM on top (Figure 8b) the linear separability improves a little bit, we can see how the majority of the class 1 points are below the separating hyperplane and the opposite happens with the points of class 0. Still, there is margin for improvement. Finetuning the model (Figure 8c) leads to a quite accurate separation,but we still have some noise on the separating hyperplane. In the last iteration, using data augmentation (Figure 8d), we obtain the most linearly separable case obtaining a 0.93 accuracy score using a linear classifier, which is a reasonable result.

In general, we can state that **training from zero** is not a good idea since we need much more epochs to converge to a solution and, if the network has a great number of parameters, it could lead to overfitting. The representations obtained by *t-SNE* get more linearly separable when we use **transfer learning with finetuning**, getting to its highest accuracy when data augmentation is added.