Statistical Methods for Machine Learning - Neural Networks

Can new architectures reach public SOTA?

Edoardo Federici*

*edoardo.federici@studenti.unimi.com

Abstract

Since 2017, Transformers have revolutionized everything Deep Learning related. First applied to text (e.g., BERT-like, T5-like, and GPT-like architectures), then successfully to images (e.g., ViT, Swin Transformer, DeIT etc.), and even on graphs (e.g., Graph Transformer), CNNs have been left behind in terms of performance. With ConvNeXt and other new architectures, the research community has tried to apply to ResNets the secret sauce that characterize Transformers, reaching state-of-the-art on ImageNet22k with all-CNN-based NNs. Can these new architectures help us reach SOTA on the Cats vs. Dogs Image Classification dataset?

Fig. 1: An oil painting by Brueghel the Elder of cats and dogs (*DALL-E 2*)

Image Classification

Image classification is the task of assigning to an input image one or more labels from a fixed set of classes. The classes may be objects, people, scenes, etc. Given that, the output space is a set made of mutually exclusive classes $\mathcal{Y} = \{1, 2, \cdots, C\}$. If there are only two labels, it is called **binary classification**, so any output must be $y \in \{0, 1\}$.

The task can be seen as estimating a function from images to labels:

$$f : \mathbb{R}^{W \times H \times K} \to \mathcal{Y} \tag{1}$$

Where $\mathcal{Y}$ has cardinality C, the number of classes.

A brief discussion

Pioneering work on deep learning for image recognition was done since the 80s, first 2010s, but we are currently in a florid period, with papers and architectures that tackle this problem being constantly released, even with new studies that can speed up training - Swin V2 (4) - or improve architectures - EfficientNetV2 (6). We can note that today the same model that took 3 days to train on a workstation years ago (MNIST with LeNet) can now be trained in less than a minute (PyTorch

(2) can even run natively on M1/M2; with a T4 LeNet on MNIST can be trained in a minute). Language models can reach into trillion parameters (Google just released a 1.6 Trillion parameters Switch Transformer, with better speed than T5-XXL (5)), and Image Models into billion parameters. New hardware can scale up architectures: more parameters equal a more robust performance, more clean data equal a more robust performance and training time can be cut with newest libraries like *JAX* (3).

A new paradigm shift is happening, and *adapting* the needed model to our task most of the time it's better (With deep-learning Image-Text models). It has been shown that pre-trained models can be *adapted* to tackle other tasks and generalize quite well. We can fine-tune BERT, a more robust DeBERTa, T5, GPT-3, Vision Transformers and so on, with a small amount of data.

Transfer Learning

Given what we said earlier, it has been shown that, given two models, where first model is trained only on a small dataset of images, while the second model is first trained on a large dataset of images and then fine-tuned on a small dataset, the second model outperforms the first, suggesting that the structural similarity between the task of interest and other data-rich tasks can be leveraged to improve performance on data-poor tasks.

Transfer learning is a technique that allows us to use a pre-trained model on a new dataset. More precisely, we first train a model on a large source dataset (with text it may be unlabelled - MLM Objective, with images large labelled corpus are available - ImageNet1k, ImageNet22k) (11). We then use this model to initialize a new model on the small labeled target dataset of interest.

Fine-Tuning

If we have a pre-trained classifier, $p(y|x, \theta_p)$, such as a ResNet (10) (or in our case more robust models), this model is trained on $x \in \mathcal{X}_p$ (images) and outputs $y \in \mathcal{Y}_p$ (e.g., the set of all labels ok ImageNet22k).

A *task* $\mathcal{T}$ can be seen as a Label Space ($\mathcal{Y}$) with a conditional probability distribution $p(x, y)$ (we can rewrite this as our predictive function), that is learned from the training data. Now we want to switch task and create a model $q(y|x, \theta_q)$ that works well for our new inputs $x \in \mathcal{X}_q$ and outputs $y \in \mathcal{Y}_p$ (e.g., cats vs dogs). In this case the probability distribution will be $q(x, y)$, and this is usually different than p.

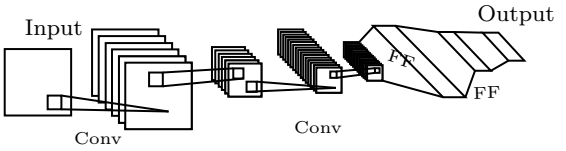
We will assume that the set of possible inputs is the same (RGB images - RGB images), so our model can be the same, having the same input. Domain $\mathcal{D}_p$ is similar to domain $\mathcal{D}_q$.

Architectures

We'll use three architectures: a modernized version of **LeNet**, a **Vision Transformer** and we implement a **ConvNeXt** (9; 8; 13) model learning to map pre-trained weights to our model and loading them.

LeNet

If we use a MLP on image data it can work, but its not good. Image can be thought as a vector, but it's a grid of pixels. This become quickly untractable if we scale up to bigger images, but this don't even exploit spatial correlation between pixels. In 1998 the LeNet-5 was introduced using two conv layers (5×5), two pooling layers (2×2) followed by two fully connected layers. In 1998 LeNet achieved SOTA accuracy on MNIST.



ViT

Vision Transformers are an adaptation of the Transformer architecture to images. Since Transform ers are a permutation equivariant architecture, if we permute the input the output is the same. Having to deal with sequences, we add positional encoding. So why not split images into patches and feed them? Every patch is seen as a *token*, and projected in the feature space. We add to this encoder a classification head and we can classify images. Transformers are *data hungry*, so it's not possible to train them well on small datasets. Pre-training costs are high (big architecture, millions of high quality images) so we fine-tune those models on our data.

In the paper they show that with fewer image inductive biases than ResNets, they perform well when trained on JFT-300M dataset.

- The first layer of the Vision Transformer linearly projects the flattened patches into a lower-dimensional space

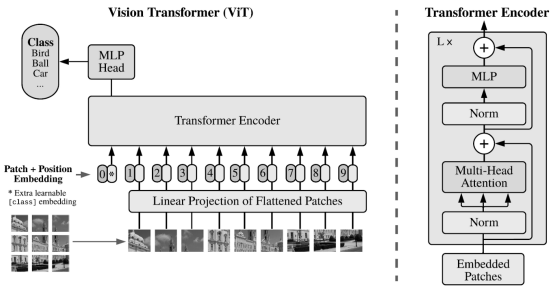


Fig. 2: Vision Transformer

- After the projection, a learned position embedding is added to the patch representations. The model learns distance within the image in the similarity of position embeddings, i.e. closer patches tend to have more similar position embeddings.
- Self-attention allows ViT to integrate information across the entire image. This *attention distance* is analogous to receptive field size in CNNs.

ConvNeXt

ConvNeXt models are a mix of ResNets and Vision Transformers. The paper shows how incrementally adding Transformers parts increase performance. They gradually modernized a standard ResNet toward the design of a vision Transformer (a Swin Transformer), reaching SOTA results on ImageNet. In july SLaK (more convnets in the 2020s), scaling up ConvNeXt kernels, performed slightly better than ConvNeXt models.

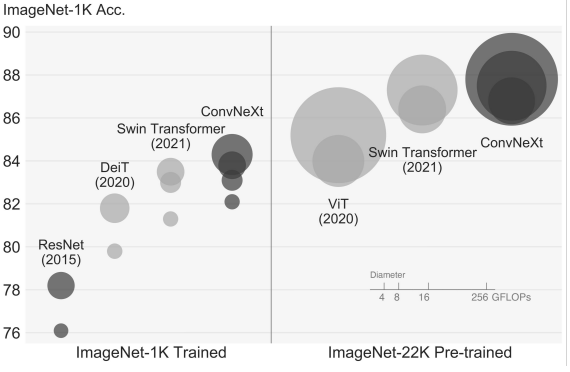


Fig. 3: Various Performances

They changed the stage compute ratio, from $(3, 4, 6, 3)$ in ResNet-50 to $(3, 3, 9, 3)$, since Swin Transformers adopted $1, 1, 9, 1$, and performance improved from 78.8% to 79.4%. They then replaced the ResNet stem cell with a patchify layer implemented using a 4×4, stride 4 convolutional layer. The accuracy changed from 79.4% to 79.5%.

They used *depthwise convolution*. Standard convolution uses a filter of size $H \times W \times C \times D$, depthwise convolution is a semplification, first convolving each input channel by a corresponding 2d filter $\mathbf{w}$, mapping the C channels to D channels with a 1×1 convolution $\mathbf{w}^{'}$:

$$z_{i,j,d} = b_d + w^{'}_{c,d} \sum_{c=0}^{C-1} (\sum_{u=0}^{H-1} \sum_{v=0}^{W-1} x_{i+u,j+v,c} w_{u,v}) \tag{2}$$

This reduces the number of FLOPs and accuracy, so, similarly to ResNeXt (7), they matched Swin Transformer width (from 64 to 96), reaching 80.5%.

Inverted bottleneck is used, as described before. This reduces FLOPs and increases performance (from 80.5% to 80.6%). In the ResNet-200, this increase performance from 81.9% to 82.6%.

With increased *Kernel Sizes*, after experiments, from 3×3 to 7×7, performance is pushed again.

They then replaced *ReLU* with *GeLU*, used *Layer Normalization* instead of batch normalization, removed some activation function (one per block left) and used fewer normalization layers (one *BatchNorm* before conv 1×1).

Optimizers

Overfitting can frequently occur when we train neural networks. Generalization is important to obtain good performance on test data.

A model is overfitting when it performs well on the training data but poorly on the test data. This can happen for a variety of reasons, including:

- The model is too complex and is learning details that do not generalize to new data.
- The model is not complex enough and is not learning the underlying patterns in the data.
- The data is too noisy and the model is picking up on random fluctuations that do not generalize to new data.

Overfitting can be prevented by using techniques such as regularization (Dropout, L1, L2, Data Augmentation etch.), early stopping, and cross-validation. Here we discuss about L2 regularization vs. Weight Decay regularization, since it is done inside our optimizer. Since *AdamW* (12) is the

Algorithm 1 Adam (L2)

given: $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 108$, $\lambda \in \mathbb{R}$
Initialize: time step $\theta \leftarrow 0$ parameter vector $\theta_{t=0} \in \mathbb{R}$, first moment vector $m_{t=0}$ second moment vector $v_{t=0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
for $t = 1$ **to** T **do**
    Randomly select mini-batch
    Compute gradients: $\nabla_\theta \mathcal{L}(\theta_{t-1})$
    $g_t \leftarrow \nabla_\theta \mathcal{L}(\theta_{t-1}) + \lambda\theta_{t-1}$
    $m_t \leftarrow \beta_1 m_{t-1} + (i - \beta_1) g_t$
    $v_t \leftarrow \beta_2 v_{t-1} + (i - \beta_2) g_t^2$
    $\tilde{m}_t \leftarrow \frac{m_t}{(1-\beta_1^t)}$
    $\tilde{v}_t \leftarrow \frac{v_t}{(1-\beta_2^t)}$
    $\eta_t = Scheduler(t)$
    $\theta_t \leftarrow \theta_{t-1} - \eta_t(\alpha\tilde{m}_t / (\sqrt{\tilde{v}_t} + \epsilon))$
end for

Algorithm 2 AdamW (WD)

given: $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 108$, $\lambda \in \mathbb{R}$
Initialize: time step $\theta \leftarrow 0$ parameter vector $\theta_{t=0} \in \mathbb{R}$, first moment vector $m_{t=0}$ second moment vector $v_{t=0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
for $t = 1$ **to** T **do**
    Randomly select mini-batch
    Compute gradients: $\nabla_\theta \mathcal{L}(\theta_{t-1})$
    $g_t \leftarrow \nabla_\theta \mathcal{L}(\theta_{t-1})$
    $m_t \leftarrow \beta_1 m_{t-1} + (i - \beta_1) g_t$
    $v_t \leftarrow \beta_2 v_{t-1} + (i - \beta_2) g_t^2$
    $\tilde{m}_t \leftarrow \frac{m_t}{(1-\beta_1^t)}$
    $\tilde{v}_t \leftarrow \frac{v_t}{(1-\beta_2^t)}$
    $\eta_t = Scheduler(t)$
    $\theta_t \leftarrow \theta_{t-1} - \eta_t(\alpha\tilde{m}_t / (\sqrt{\tilde{v}_t} + \epsilon)) + \lambda\theta_{t-1}$
end for

'default' optimizer for deep neural networks today (BigScience's BLOOM, DALL-E, CLIP, DeBERTa etc.) reading the paper it said how on our task it's better than standard Adam (1). In the paper they discuss that, even if Adam is efficacely used for RNNs and Feed-Forward nets, sota-results for Image Classification are achieved by applying SGD with Momentum. They found that:

Proposition 1 *Weight decay = L2 regularization for standard SGD.*

Proposition 2 *Weight decay ≠ L2 regularization for adaptive gradients.*

Their experiments demonstrate that decoupling weight decay and loss-based gradient updates leads to improved generalization for a range of tasks and architectures, so they proposed their own version of Adam, **AdamW**.

They then empirically show that their version of Adam - with decoupled weight decay - yields substantially better generalization performance than Adam with L2 regularization.

Experiments

The experiments were done on an NVIDIA 3060 12GB OC and Colab's T4.

LeNet

Dropout was added before the bigger dense layer, which helps reducing overfitting.

5-fold, SGD Momentum

For the first part of the experiment the zero-one loss is computed for LeNet model, using zero_one_loss over the five folds. SGD with momentum is used with a standard momentum of 0.9 and a initial learning rate of $1e-3$. We report the results in this table:

Table 1. SGD Momentum results

fold	zero_one loss	val_acc	train_loss
1	0.2015	0.7985	0.6353
2	0.2013	0.7987	0.4712
3	0.2118	0.7883	0.5838
4	0.2210	0.7790	0.4397
5	0.1977	0.8022	0.6226

Which results in an overall score of 0,2067.

5-fold, Adam

Here we use Adam with initial learning rate of $1e-3$. In our experiments this is the optimizer that performs better with our data. This can be due to sub-optimal hyperparameters choice for training (even though AdamW should adapt to a broader range of parameters and perform better in this case).

Table 2. Adam results

fold	zero_one loss	val_acc	train_loss
1	0.1738	0.8263	0.5076
2	0.1860	0.8140	0.5354
3	0.1789	0.8213	0.4723
4	0.1933	0.8067	0.3835
5	0.1850	0.8149	0.4804

Which results in an overall score of 0,1834.

5-fold, AdamW

Here we use AdamW with initial learning rate of $1e-3$ and initial weight decay $5e-5$. We can see that AdamW performs better than SGD with momentum, but is comparable to Adam results. Weight decay can reduce overfitting, but it was not investigated during the experiments. It does not improve results in our experiment.

Table 3. AdamW results

fold	zero_one loss	val_acc	train_loss
1	0.1808	0.8192	0.5076
2	0.1798	0.8202	0.3439
3	0.1863	0.8138	0.6052
4	0.2003	0.7997	0.4455
5	0.1838	0.8162	0.5485

Which results in an overall score of 0,1862.

Vision Transformer

Here we use a straightforward approach with a pre-trained base model (google/vit-base-patch16-224-in21k), trained on ImageNet 1k. We fine-tune it for five epochs with AdamW. Vision Transformers are trained in JAX, so tensorflow ported weights might perform slightly worse. Overall results can be found on the table below. It reaches 0.9942 after one epoch so it's quite good.

ConvNeXt

Firstly we computed mean and std dev over our dataset, which turned out to be mean - $[0.49, 0.46, 0.42]$, std - $[0.26, 0.25, 0.25]$. Tensorboard was used to monitor metrics.

ConvNeXt

With ConvNeXt models we are comparing training the model (ConvNeXt Small and Base) from scratch on our dataset. Adam is used to train them and fine-tuning them on Cats vs. Dogs. Data was splitted in train and test set, with sizes 0.8, 0.2 respectively.

ConvNeXt-S

Cats vs. Dogs Training:

- `batch_size: 64`, `lr: 2e-3`

We start our analysis with the small model. After training the model for 30 epochs it can be seen that the model doesn't overfit at all, leaving room for further improvements. It reaches an accuracy of 72 % after 30 epochs.

ImageNet22k pre-trained, fine-tuning:

- `batch_size: 64`, `lr: 4e-5`

So we perform the same experiments with the pre-trained one and it outperforms the first model. On the first epoch we note an accuracy of 0.98 % on the validation set, which is quite impressive. These models need a lot of data, vision-transformers especially, and the design is adapted for larger and larger trainings. ImageNet has millions of images, here we deal with 25k, 22k if we subtract the evaluation set. After 10 epochs validation loss is still slowly decreasing

ConvNeXt-B

Cats vs. Dogs Training:

- `batch_size: 32`, `lr: 2e-3`

This is quite over-powered for our dataset. Training the Base model on Cats vs. Dogs requires more compute time. With 6 hours on training it reaches an accuracy of 0.8949 but because I stopped it.

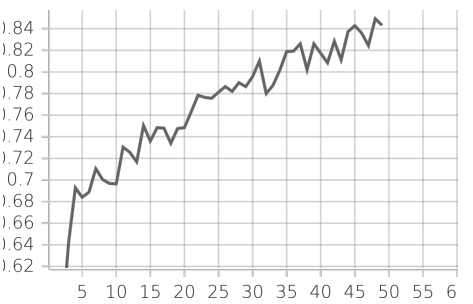


Fig. 4: Validation accuracy for the base model trained for 50 epochs. Validation loss is decreasing also

ImageNet22k pre-trained, fine-tuning:

- `batch_size: 32`, `lr: 4e-5`

The pre-trained one after one epoch shows an accuracy of 0.9832 which is promising. With other images we can reach further (we can enlarge our dataset with web search, as an example). After only few epochs we reach higher scores. This model is the one that performs better, so we used Huggingface optimized Trainer and Their checkpoint mapping to see if we had a performance decrease. It does, after the first epoch we note an incredible accuracy of 99.80%. After the second 99.89%. After five: 99.96 %. How better than this? After the seventh epoch it slowly starts to overfit, validation loss stopped decreasing and accuracy is not improving anymore. The default 5 epochs are enough. I used torch implementation of AdamW and parameters are slightly different. Also to be noted is the learning rate scheduler used in the Trainer that stabilizes convergence.

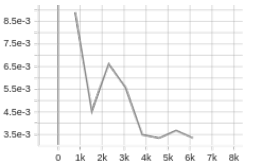


Fig. 5: Val loss

Fig. 6: Val accuracy

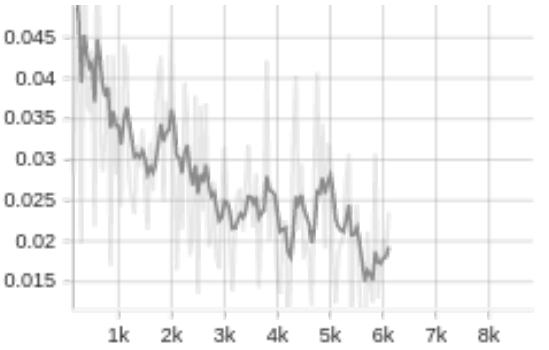


Fig. 7: Training loss. It's quite noisy since we are dealing with smaller ranges

Table 4. Results

model	train_loss	val_acc	epochs
ViT-B	0.0042	0.9972	4
ConvNeXt-T-22k	0.0320	0.9888	6
ConvNeXt-T	0.5183	0.7424	30
ConvNeXt-B-22k-1k	0.0267	0.9932	9
ConvNeXt-B	0.1003	0.8492	50
ConvNeXt-B-22k-1k-HF	0.0201	0.9996	5

Results of the various models trained for Image Classification. Only best model, with its corresponding epoch and loss, is showed.

- epoch = 8.0
- eval_accuracy = 0.9996
- eval_loss = 0.0034
- eval_samples_per_second = 134.512

Conclusion

Overall, the results of this image classification study suggest that modern architectures perform very well when pre-trained weights are adapted using fine-tuning. The convolutional neural networks employed in this research were able enough to learn high-level features from the dataset that allowed for effective classification of the images. Furthermore, the results also indicate that even though there is some variability in the performance of the models across different runs, a simple modernized CNN is robust and can achieve consistent results. But newer models when fine-tuned correctly can achieve higher results and even outperforms previous approaches like Vision Transformers.

There are a few possible directions for future work. Tuning hyperparameters can improve results. One could even try to improve the classification accuracy by increasing the size of the dataset. Studies suggest that image classification pre-training is more

Table 5. ConvNeXt results on the original kaggle dataset: https://paperswithcode.com/sota/image-classification-on-cats-vs-dogs.

	Public SOTA (ViT)			Our fine-tuned ConvNeXt		
Model	loss	acc	epochs	loss	acc	epochs
Comparison	0.0182	0.9937	5	0.0103	0.9973	5

Results can be found on paperswithcode

effective when images are poorly labeled, so a two-step approach (training on scraped images and fine-tuning on labelled samples) could improve the result (which is already quite high).

Acknowledgments

I/We declare that this material, which I/We now submit for assessment, is entirely my/our own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my/our work. I/We understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me/us or any other person for assessment on this or any other course of study.

References

1. Kingma, Diederik P. and Ba, Jimmy. Adam: A Method for Stochastic Optimization. *arXiv*, 2014.
2. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *arXiv*, 2019.
3. JAX: composable transformations of Python+NumPy programs. *arXiv*, 2018.
4. Liu, Ze and Hu, Han and Lin, Yutong and Yao, Zhuliang and Xie, Zhenda and Wei, Yixuan and Ning, Jia and Cao, Yue and Zhang, Zheng and Dong, Li and Wei, Furu and Guo, Baining. Swin Transformer V2: Scaling Up Capacity and Resolution. *arXiv*, 2022.
5. Colin Raffel and Noam Shazeer and Adam Roberts and Katherine Lee and Sharan Narang and Michael Matena and Yanqi Zhou and Wei Li and Peter J. Liu. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *arXiv*, 2020.
6. Tan, Mingxing and Le, Quoc V. EfficientNetV2: Smaller Models and Faster Training. *arXiv*, 2021.
7. Wenfeng Feng, Xin Zhang, Guangpeng Zhao. ResNetX: a more disordered and deeper network architecture. *arXiv*, 2019.
8. Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *arXiv*, 2020.
9. LeCun, Y. and Boser, B. and Denker, J. S. and Henderson, D. and Howard, R. E. and Hubbard, W. and Jackel, L. D. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1989.
10. Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. Deep Residual Learning for Image Recognition. *CoRR*, 2015.
11. Olga Russakovsky and Jia Deng and Hao Su and Jonathan Krause and Sanjeev Satheesh and Sean Ma and Zhiheng Huang and Andrej Karpathy and Aditya Khosla and Michael Bernstein and Alexander C. Berg and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 2015.
12. Loshchilov, Ilya and Hutter, Frank. Decoupled Weight Decay Regularization. *arXiv* 2017.
13. Zhuang Liu and Hanzi Mao and Chao-Yuan Wu and Christoph Feichtenhofer and Trevor Darrell. A ConvNet for the 2020s. *arXiv* 2022.