Neural Networks & Representation Learning (CS-587)
Assignment #3

Due on May 24, 2023

Ass. Prof. N. Komontakis

University of Crete
Department of Computer Science

Nikolaos Kougioulis (ID 1285)

May 24, 2023

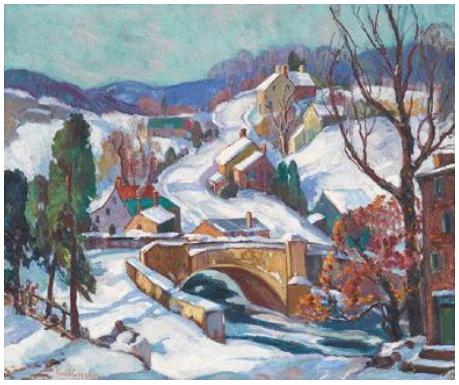
Assignment 2a

The first part of the Assignment is concerned with the task of *Transfer Learning*. Transfer learning refers to the use of a pre-trained model on a different task that the one it was originally trained for. In this context, the model exploits the knowledge acquired from that previous task to improve the generalization about another task [1].

In this assignment, we start with AlexNet [2], a convolutional neural network (cNN) model consisting of 8 layers, trained on over a million images to classify them on 1000 classes. Our task is to perform transfer learning to classify paintings on a range of 10 styles (baroque, romantism etc), by obtaining a sample from WikiArt[3] containing 4000 paintings using the provided `.py` mass downloader script.

(a) Neoclassicism

(b) Romantism



(c) Impressionism

(d) Expressionism

Figure 1: Four paintings from the WikiArt dataset with their respective class.

The architecture of AlexNet consists of convolutional, pooling, and FC layers. Starting with an input layer of 224x224x3 (224x224 dimensions and 3 channels representing the RGB spectrum). The first convolutional layer applies a filter of size 96 with stride, extracting low-level features like as edges and corners. After

[1] https://www.tensorflow.org/tutorials/images/transfer_learning

[2] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). *ImageNet classification with Deep Convolutional Neural Networks.* In Advances in Neural Information Processing Systems (pp. 1097-1105).

[3] https://www.wikiart.org/en/paintings-by-style

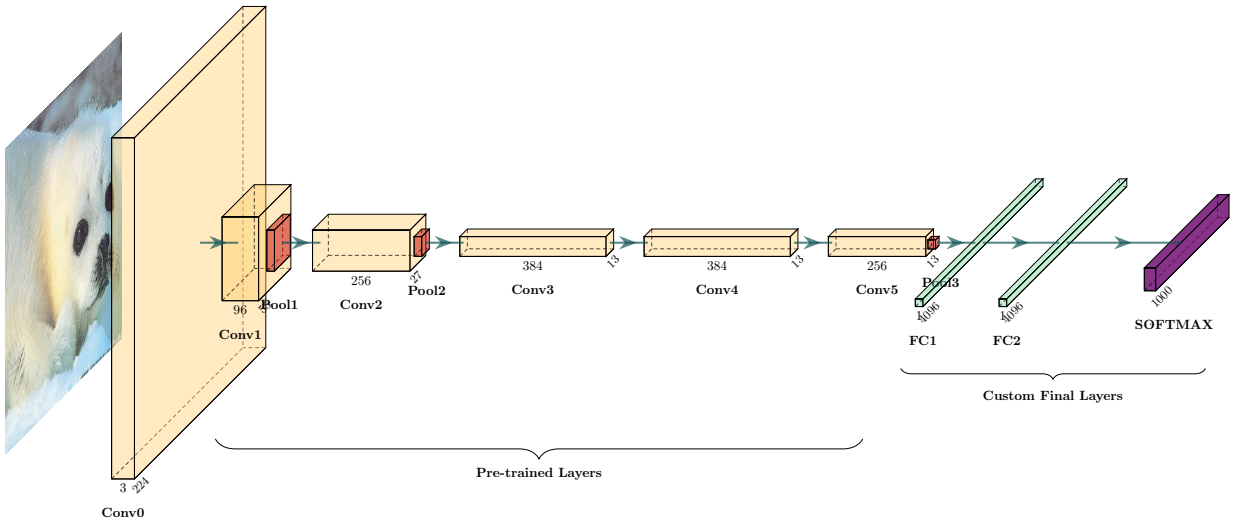


Figure 2: Illustration of the architecture of AlexNet, a deep convolutional neural network (cNN) proposed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton in 2012. The architecture of AlexNet gained significant attention and became a breakthrough model in the field of computer vision when it won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012, significantly outperforming the state-of-the-art methods of that period. In the task of transfer learning, the final FC layers (denoted with an underbrace and noted *custom final layers*) are re-trained from scratch (fine-tuned) to capture the higher-level features of our feature-specific dataset, while vastly reducing the coputational costs of training a complete cNN architecture from scratch. Note that in our implementation on WikiArt, since we are dealing with 10 classes, the 1000 logits in the final layer are being replaced with 10 logits (`num_classes=10`).

each of the first two convolutional layers, a max pooling layer with filter and stride is applied, hence reducing spatial dimensions and aiding in capturing invariant features. Local Response Normalization layers are applied between convolutional pooling layers to enhance generalization. The subsequent conv layers employ smaller filters (5x5 and 3x3) to capture more complex patterns and higher-level features. At the deepest level, following the convolutional and pooling layers, three fully connected (FC) layers of 4096 neurons each are present, with the final fully connected layer (output layer) having 1000 neurons (number of classes in the ImageNet dataset) and outputting the respective unnormalised (1000) logits. The softmax activation function computes the probabilities of each class given the input image using the logits acquired previously from the FC layer. An illustration of the AlexNet architecture is shown in Figure 2.

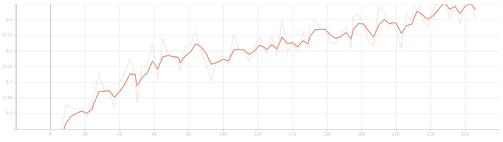
An important aspect of transfer learning is fine-tuning. In this case, a selection of the final fully connected layers are removed from the pre-trained model and re-trained from scratch. Unfreezing the top layers refers to making those layers trainable so that their weights can be fine-tuned based on the specific task or dataset. As the network begins by extracting generalized low-level features of the data and proceeds to more specific and dependent characteristics as the layers get deeper, this allows the model to keep low-level representations and be re-trained from scratch on the specific features of the dataset, which in practice is smaller and with a more specialized context that the one the pre-trained model has been trained on.

We begin by automatically obtaining a total of 4000 paintings from WikiArt, using the accompanied script, resulting in a 1.7 GB folder (located at `Utilities/data/images`), and split into a training and a validation set (paths of images on the files `Utilities/data/train.txt` and `Utilities/data/test.txt`), and then obtaining the weights of the pre-trained AlexNet model on ImageNet (`Weights/bvlc_alexnet.npy`).

Learning parameters are left as default, that is, `learning_rate = 0.01`, `num_epochs = 10`, `batch_size = 128` and `dropout_rate = 0.5`, `num_classes = 10`. By setting in the `train_layers` list the name of the layers we are interested in un-freezing and re-training (and hence skipping the layers that are not named in the list), we can remove for example the last FC layer (*FC2* in Figure 2) or the last two FC layers (*FC2* and *FC1* in Figure 2) and re-training by fine-tuning them.

We obtain the defined summaries (cross-entropy loss and accuracy) every `display_step = 3` batches and plot them using Tensorboard[4]. By unfreezing the last fully-connected (FC) layer, a validation accuracy of `0.39062` while by unfreezing the two final FC layers a validation accuracy of `0.40625`.

The training accuracy and cross-entropy loss of each of the two fine-tuning cases mentioned previously are illustrated in Figures ?? and ??, while the histogram and distribution of the gradients of the biases and weights of the re-trained FC layers are available in the Appendix. Notice how the plots are a bit smoother when two FC layers are unfreezed and retrained.



(a) Training accuracy

(b) Training cross-entropy loss

Figure 3: Training Accuracy and loss on the un-freezed AlexNet with the final FC layer (`fc8`) re-trained, using Tensorboard.

When we remove and retrain the last FC layer of the network, it is probable that the network may struggle adapting to the new task, resulting in a lower accuracy. This happens since the final FC layer maps the learned features to the specific classes of the original ImageNet dataset (two orders of magnitude lower and of different context), which may not directly correspond to the feature representation of the different classes in the WikiArt dataset. This happens because the earlier layers in a deep neural network, like the convolutional layers, are responsible for learning general features, while the later FC layers tend to capture more specific and task-dependent information of the data.

Similarly, when the last two FC layers are un-freezed and re-trained, although the network learns and adapts the richer high-level feature representations of WikiArt more extensively, this results in a slight improvement in accuracy, although not enough to fully capture the differences between different style images in the unknown test set. Of course, this stems from the degree of similarity between Imagenet and the target dataset (in our case, WikiArt).

In more detail, a poor performance of the pre-trained AlexNet on WikiArt can be explained further by the following points:

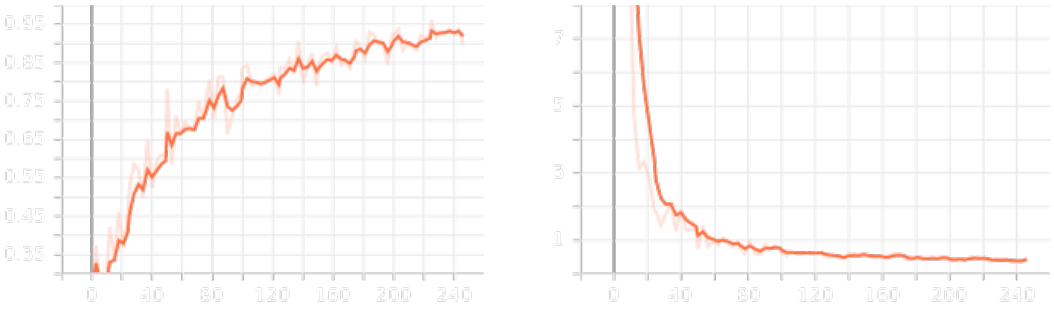
I *Textures & colors*: Paintings differ than natural images due to their textures that are dependent on the painting technique used, the material of the brushes, the stroke, the personal style of the painter, the

[4] https://www.tensorflow.org/tensorboard

color palette used etc. These combinations can significantly vary between different art movements and styles, thus making the pre-trained AlexNet less effective in capturing these artistic characteristics.

II *Semantic Meaning*: Paintings are created to convey various artistic concepts, emotions, and narratives, which might not align with the object-centric semantics learned by AlexNet from ImageNet. The pre-trained model may not have learned the specific visual cues and symbolic representations relevant to the artistic concepts present in the WikiArt dataset.

III *Classes*: The ImageNet dataset consists of various categories including a plethora of species and breeds of animals (250: siberian husky, 265: toy puddle etc), items (514: cowboy boot, 530: dining clock, 849: teapot etc) and vehicles (561: forklift, 751: race car, 864: tow car etc). On the other hand, paintings in the WikiArt dataset are focused on specific art movements, genres, or themes, hence the pre-trained AlexNet might not have learned discriminative features for these specific painting-related themes.



(a) Training accuracy (b) Training cross-entropy loss

Figure 4: Training Accuracy and loss on the un-freezed AlexNet with the final two FC layers (`fc7`, `fc8`) re-trained, using Tensorboard.

Assignment 2b

For part B, the task is to visualize the saliency map of various input images, using the VGG16 architecture and the Vanilla Gradient method.

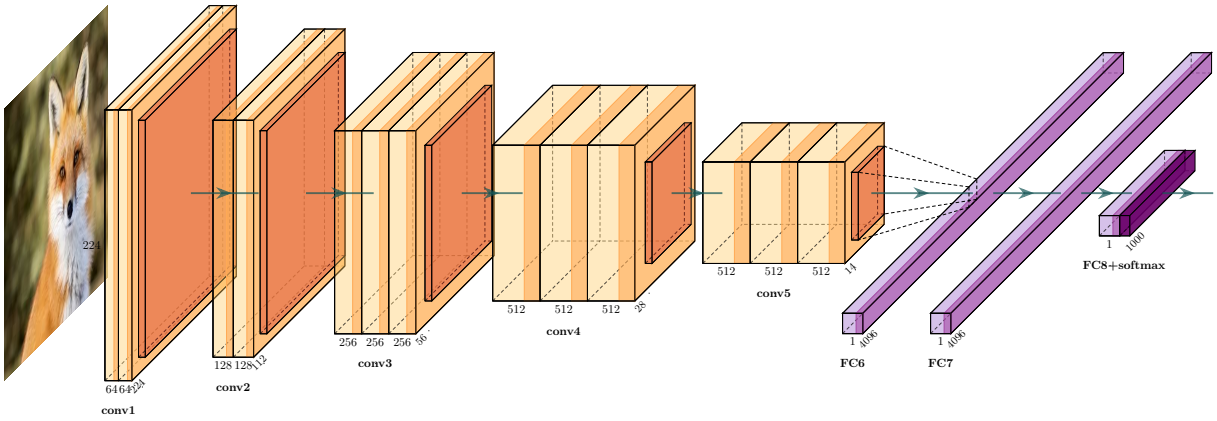


Figure 5: Illustration of the architecture of VGG16. Like before, *redfox* corresponds to class 277 in ImageNet.

VGG16 (Visual Geometry Group 16) is a convolutional neural network (CNN) architecture developed by the Visual Geometry Group at the University of Oxford[5], that gained significant attention for its impressive performance in the 2014 ILSVRC challenge. Trained on a subset of ImageNet with 1000 classes, similarly to AlexNet, as the name suggests, VGG16 is comprised of 16 layers (13 convolutional layers and 3 fully connected layers).

The architecture utilizes 3x3 convolutional filters consistently throughout the network as well as implementing max-pooling layers and finally a softmax activation function, like AlexNet. See Figure 5 for an illustration of the architecture of the model.

The purpose of *saliency maps* is to serve as a visualization technique, offering valuable understanding into the specific reasoning behind a deep learning model's decision (what the model "sees") on the classification of an image. Saliency maps are rendered as heatmaps of the original image, where intense colors represent regions that have a big impact on the model's decision and are more attention-grabbing than the remaining parts of the image. It allows us to obtain a behind-the-curtain understanding on what high-level features of the input influence the final classification decision but also to distinguish which features lead to incorrect decisions[6].

We create the saliency map, first of the flamingo and then of the three images in the `MyImages` folder, after classifying them on the VGG16 network, using the original saliency map algorithm in supervised deep learning introduced by Simonyan et. al. in 2013[7], known as *Vanilla Gradient Mask* [8]. Vanilla Gradient has

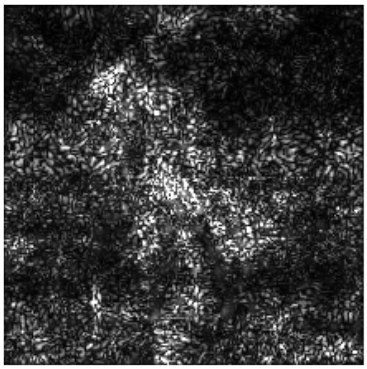
[5]Simonyan, K., & Zisserman, A. (2014). *Very Deep Convolutional Networks for Large-Scale Image Recognition.* arXiv preprint arXiv:1409.1556.

[6]YouTube. (2021). L13.8 *What a CNN Can See - STAT 453: Intro to Deep Learning and Generative Models*, University of Wisconsin-Madison. Available on YouTube. Retrieved May 12, 2023, from https://www.youtube.com/watch?v=PRFP5YC3u7g.

[7]Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman, *Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps*, ICLR Workshop 2014.

[8]https://andrewschrbr.medium.com/saliency-maps-for-deep-learning-part-1-vanilla-gradient-1d0665de3284

Image of a flamingo by the beach



Saliency map of the flamingo

Figure 6: Input image of a flamingo, classified by VGG16 in class 130 (class flamingo) with its vanilla gradient saliency map.

proven to be quite robust compared to other techniques, while also standing out as the simplest algorithm among gradient-based approaches. The calculation and visualization of the vanilla gradient is the following:

1. Do a forward pass of the image I through the network
2. Calculate the scores for every class C, $S_C(I) = w_C^T I + b_c$
3. Enforce the derivative w.r.t. I (i.e. the input image) of score vector S at last layer for all classes except class C to be 0, while for C set it to 1, $\text{argmax}_I S_C(I) - \lambda ||I||_2^2$
4. Back-propagate this derivative until the start, $w = \frac{\partial S_C}{\partial I}|_{I_0}$
5. Render the gradients and obtain the saliency map of I

We implement the above steps in the accompanied `.ipynb` notebook of the report and firstly obtain the saliency map of the flamingo, shown in Figure ??.

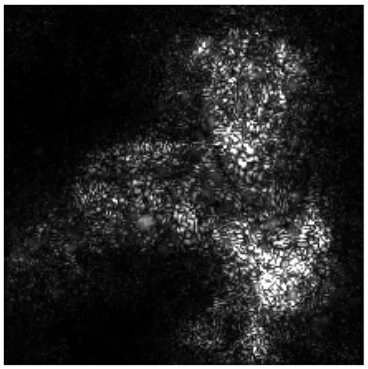
The high values around the body of the flamingo in the saliency map can be explained by the body of the flamingo being vibrant in color (pink due to their food, algae and brine shrimp) and thus highly distinct from its surroundings. High values around the clouds can be attributed to the high contrasting color between the flamingo and the sky, as well as the environment in which an image of a flamingo is often captured.

We then repeat the process for the rest of the images located in the folder `My-Images`, a doberman (class 236), a teddy bear (class 850) and an image of a cat (class 281). The resulting saliency maps for each of the three images in the aforementioned folder are shown in Figure 7, after being classified correctly by the model. Some explaining on the resulted saliency maps for the three images is the following:

- *Dog:* The highlighting of the crossed legs and all-around shape suggests that they provide cues about the dog's posture and movement. The highlighting of the face, the nose-mouth region and the ears, are a crucial area for attention as it predominately characterizes the dog species.
- *Teddy bear:* Similarly to the dog, the high values around the top of the head and legs in the saliency map of the teddy bear are a result of its distinct shape and the presence of prominent features in teddy bears. The legs might also be considered visually significant as they provide information about the pose or position of the teddy bear (different than a wild bear).

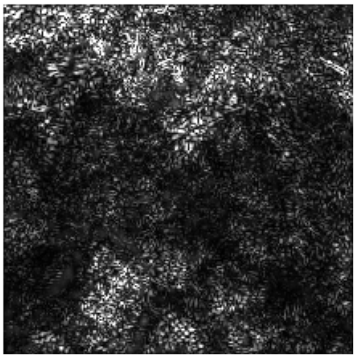
- *Cat:* Finally, the areas around the nose, eyes, ears exhibit distinct shapes and patterns present in cats (and big cats in general like leopards), with cats being differentiated from other big cats due to their smaller shape and stature, thus causing the saliency map to depict attention on the all-around shape of the cat.

(a) Doberman



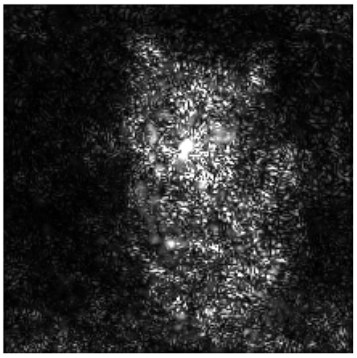
Corresponding saliency map

(b) Teddy Bear



Corresponding saliency map

(c) Cat



Corresponding saliency map

Figure 7: The three images in the `myImages` folder with their respective vanilla gradient saliency maps.

References

[1] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.

Appendix

Troubleshooting

The `.ipynb` notebooks of the assignment were run on **Ubuntu 22.04 LTS**, in **Python 3.5.6** (—Anaconda, Inc.— (default, Aug 26 2018, 21:41:56) [GCC 7.3.0] linux /home/nikolas/anaconda3/envs/cs587_hw3/bin/python), downgraded **NumPy 1.16.5** (['/home/nikolas/anaconda3/envs/cs587_hw3/lib/python3.5/site-packages/numpy'] and scikit-learn. 1.16.5), **TensorFlow 1.15**, **Pandas** and **Pickle 4.0.0** in the created `cs587_hw3` Anaconda virtual environment (`conda create cs587_hw3 python=3.5` and `conda activate cs587_hw3`). Monitoring of the model was done using Tensorboard, `tensorboard --logdir finetune_alexnet/wikiart`. The source files of both `AlexNet.py` and `VGG16.py` containing the `np.load` command were modified to include the `allow_pickle=True` argument.

One can obtain the list of packages we had to downgrade and install while trying to get the assignment running, by executing `pip list --format=freeze > requirements.txt` in the `cs587_hw3` virtual environment:

```
absl-py==0.15.0
asn1crypto==1.4.0
astor==0.8.1
bleach==3.3.1
certifi==2020.6.20
cffi==1.11.5
colorama==0.4.4
cryptography==2.3.1
cycler==0.10.0
decorator==5.1.1
defusedxml==0.7.1
entrypoints==0.2.3
gast==0.2.2
google-pasta==0.2.0
grpcio==1.41.1
h5py==2.10.0
idna==3.3
imageio==2.9.0
importlib-metadata==2.1.3
importlib-resources==3.2.1
ipykernel==4.10.0
ipython==5.8.0
ipython-genutils==0.2.0
ipywidgets==7.4.1
Jinja2==2.11.3
jsonschema==2.6.0
jupyter==1.0.0
jupyter-client==5.3.3
jupyter-console==5.2.0
jupyter-core==4.5.0
Keras-Applications==1.0.8
Keras-Preprocessing==1.1.2
kiwisolver==1.1.0
Markdown==3.2.2
MarkupSafe==1.0
matplotlib==3.0.3
mistune==0.8.3
nbconvert==5.5.0
nbformat==5.1.3
networkx==2.4
notebook==5.5.0
numpy==1.16.5
```

```
olefile==0.46
opt-einsum==3.3.0
packaging==20.9
pandas==0.23.4
pandocfilters==1.5.0
pexpect==4.8.0
pickleshare==0.7.5
Pillow==4.0.0
pip==20.3.4
prompt-toolkit==1.0.15
protobuf==3.19.6
ptyprocess==0.7.0
pycparser==2.20
Pygments==2.11.2
pyOpenSSL==18.0.0
pyparsing==2.4.7
PySocks==1.6.8
python-dateutil==2.8.2
pytz==2021.3
PyWavelets==1.1.1
pyzmq==17.1.2
qtconsole==4.7.7
QtPy==1.11.2
scikit-image==0.15.0
scipy==1.4.1
Send2Trash==1.8.0
```

Extra figures

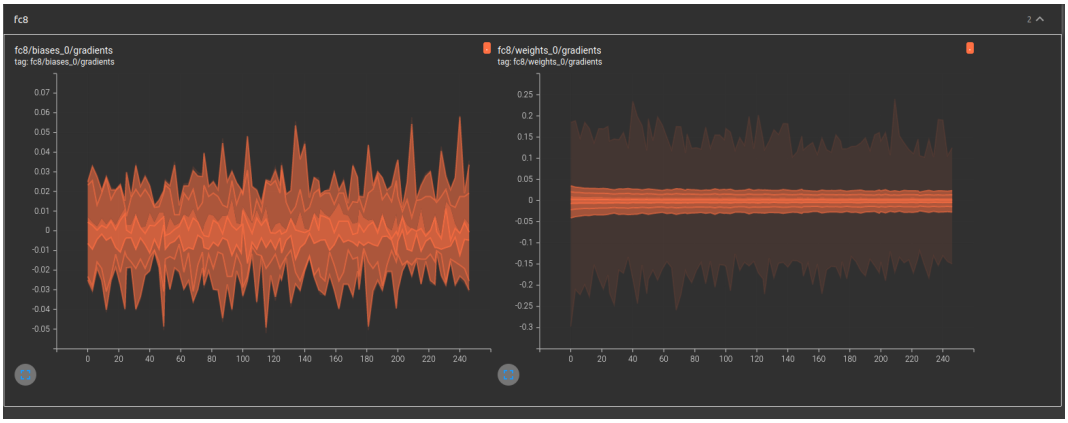


Figure 8: Gradients distribution for the biases and weights of layer FC8 of the AlexNet model with the final FC layer (fc8) removed and re-trained on WikiArt.

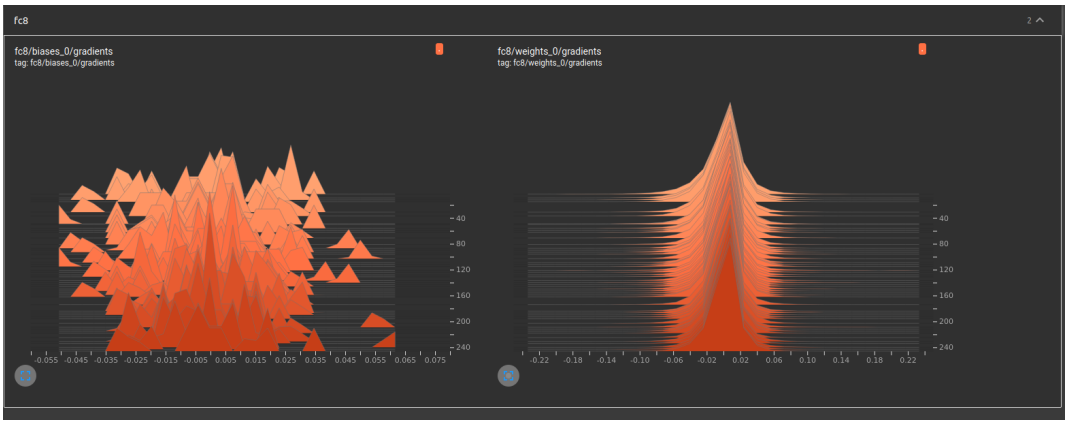


Figure 9: Histogram for the gradients of the biases and weights of layer FC8 of the AlexNet model with the final FC layer (fc8) removed and re-trained on WikiArt.

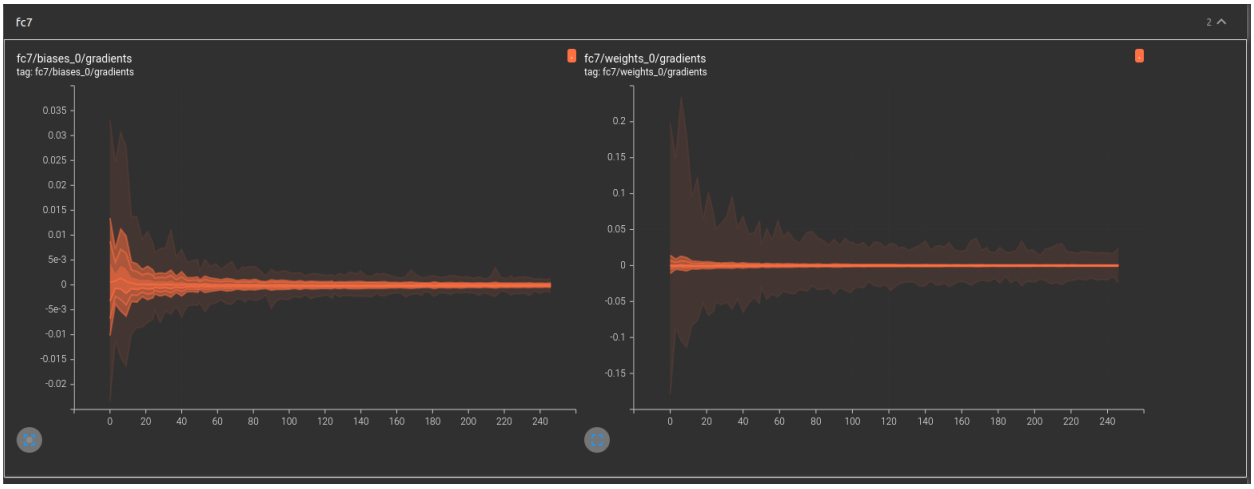


Figure 10: Gradients distribution for the biases and weights of the FC layer `fc7` of the AlexNet model with the final two FC layers (`fc7` and `fc8`) removed and re-trained on WikiArt.

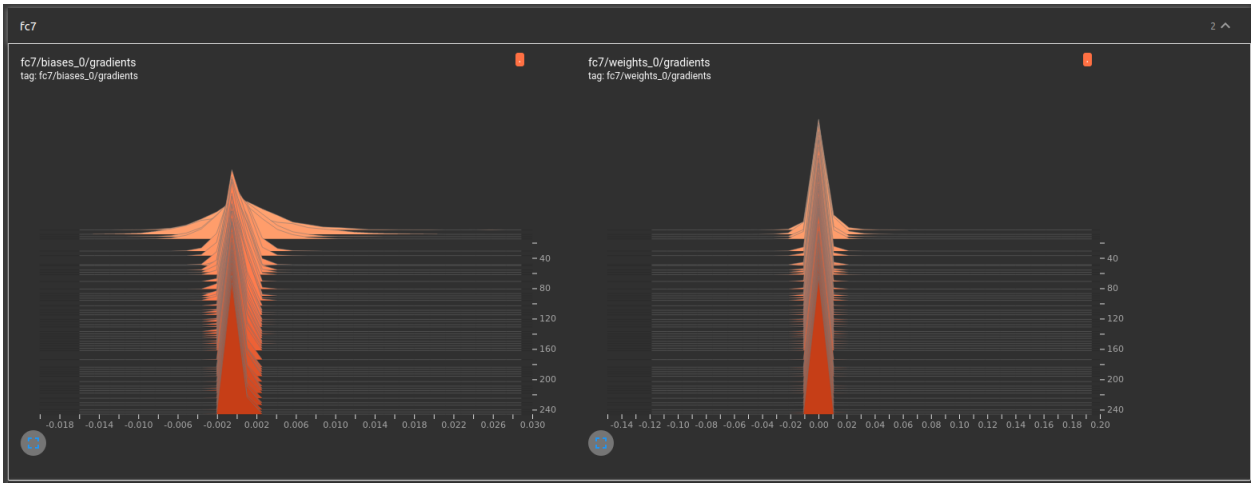


Figure 11: Histogram for the gradients of the biases and weights of the FC layer `fc7` of the AlexNet model with the final two FC layers (`fc7` and `fc8`) removed and re-trained on WikiArt.

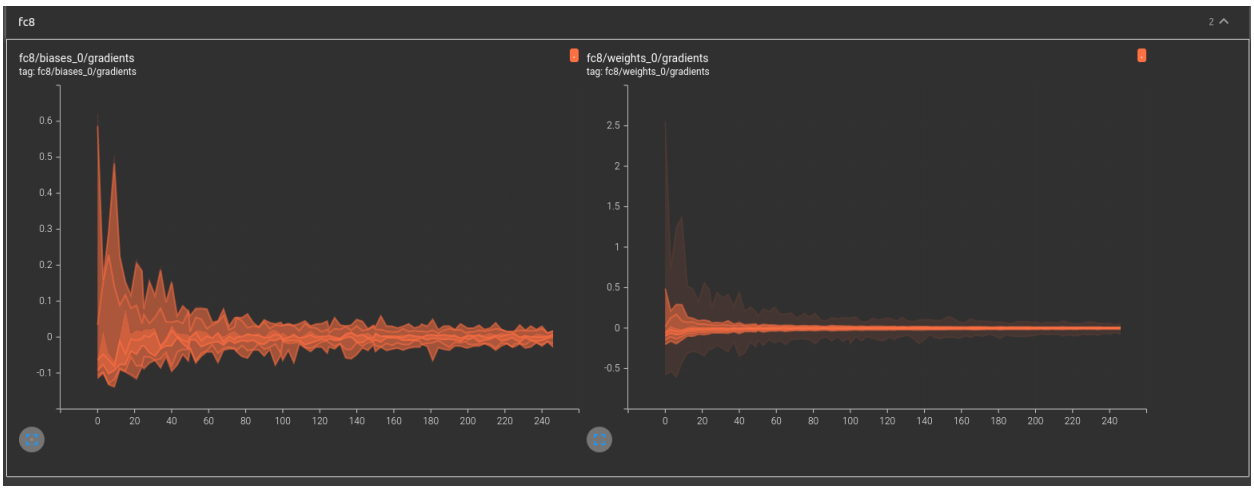


Figure 12: Gradients distribution for the biases and weights of the FC layer `fc8` of the AlexNet model with the final two FC layers (`fc7` and `fc8`) removed and re-trained on WikiArt.

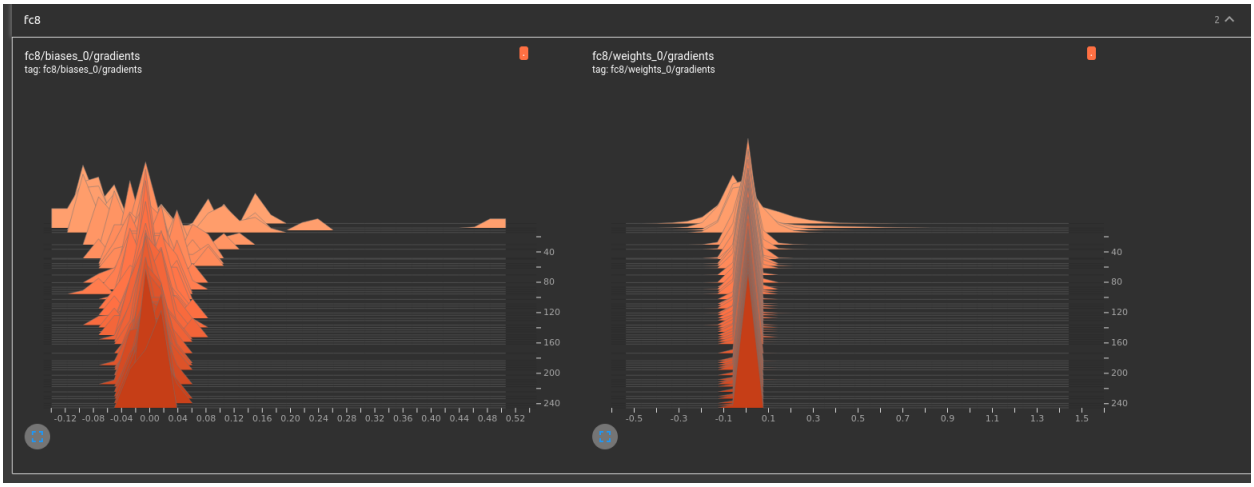


Figure 13: Histogram for the gradients of the biases and weights of the FC layer `fc8` of the AlexNet model with the final two FC layers (`fc7` and `fc8`) removed and re-trained on WikiArt.