Machine Learning
Università della Svizzera italiana

Assignment 2

Morale Mariciano Jeferson

June 7, 2024

Please refer to the **ReadMe** file to get the exact all questions. This is just a template of how your report should look like. In this assignment, you are asked to:

1. Implement a fully connected feed-forward neural network to classify images from the **Cats of the Wild** dataset.
2. Implement a convolutional neural network to classify images of **Cats of the Wild** dataset.
3. Implement transfer learning.

Both requests are very similar to what we have seen during the labs. However, you are required to follow **exactly** the assignment's specifications.

1 Image Classification with Fully Connected Feed Forward Neural Networks (FFNN)

In this task, you will try and build a classifier for the provided dataset. This task, you will build a classic Feed Forward Neural Network.

1. Download and load the dataset using the following link. The dataset consist of 7 classes with a folder for each class images. The classes are 'CHEETAH' ,'OCELOT', 'SNOW LEOPARD', 'CARACAL', 'LIONS', 'PUMA', 'TIGER'. Check Cell 1 in 'example.ipynb' to find the ready and implemented function to load the dataset.
2. Preprocess the data: normalize each pixel of each channel so that the range is [0, 1].
3. One hot encode the labels (the y variable).
4. Flatten the images into 1D vectors. You can achieve that by using [torch.reshape] or by prepending a [Flatten layer] to your architecture; if you follow this approach this layer will not count for the rules at point 5.

5. Build a Feed Forward Neural Network of your choice, following these constraints:
    - Use only torch nn.Linear layers.
    - Use no more than 3 layers, considering also the output one.
    - Use ReLU activation for all layers other than the output one.
6. Draw a plot with epochs on the x-axis and with two graphs: the train accuracy and the validation accuracy (remember to add a legend to distinguish the two graphs!).
7. Assess and comment on the performances of the network on your test set, and provide an estimate of the classification accuracy that you expect on new and unseen images.
8. **Bonus** (Optional) Train your architecture of choice (you are allowed to change the input layer dimensionality!) following the same procedure as above, but, instead of the flattened images, use any feature of your choice as input. You can think of these extracted features as a conceptual equivalent of the Polynomial Features you saw in Regression problems, where the input data were 1D vectors. Remember that images are just 3D tensors (HxWxC) where the first two dimensions are the Height and Width of the image and the last dimension represents the channels (usually 3 for RGB images, one for red, one for green and one for blue). You can compute functions of these data as you would for any multi-dimensional array. A few examples of features that can be extracted from images are:
    - Mean and variance over the whole image.
    - Mean and variance for each channel.
    - Max and min values over the whole image.
    - Max and min values for each channel.
    - Ratios between statistics of different channels (e.g. Max Red / Max Blue)
    - Image Histogram (Can be compute directly by temporarely converting to numpy arrays and using np.histogram)

    But you can use anything that you think may carry useful information to classify an image.

    N.B. If you carry out point 7 also consider the obtained model and results in the discussion of point 6.

Reasoning

Every answer provided first depict the overall plain result. Then, prompts the analytical data that lead to such result. In addition, a theoretical motivation for the outcomes is given. Finally, some metaphors are provided by me for the best I could approximate such concepts to how humans behave and interact with nature, in order to ease the understanding of these concepts.

Apparatus and Configuration

The jupyter notebook was developed using the following configurations in order to produce the most deterministic outcome at every new run.
Apparatus description:

- **CPU** Ryzen 3900XT
- **GPU** RTX 3070 8GB VRAM
- **RAM** 16 GB
- **OS** Windows 11 Pro 23H2

The first scratch of the assigment was developed on Kaggle, were the only hardware specs provided were a python environment in a notebook with a NVIDIA P100 GPU dedicated to the task, CUDA enabled during training.
the configuration description for the training and assignment overall are:

- **seed** 20020309
- **learning rate** 0.001
- **batch size**
- **cuda** enabled
- **epochs** 101
- **loss function** cross entropy
- **optimizer** Adam

The reason of using CrossEntropy as **loss function** from pytorch is that the module implements both CrossEntropy and SoftMax at every layer but the last output layer, which is exactly the desired behavior.
The training loop implemented follows a similar pattern provided by PyTorch documentation.
The seed is passed to the functions `train_test_split(..., random_state=...)` and set initially to `np.random.seed(), torch.manual_seed()` to ensure the reproducibility of the results. Since I had problems with the dataset and loaders, I figured out that by setting the seed to the same value to all "pseudo-random" functions, I will eventually have the same charateristics of the dataset, i.e. same ordering. It was a nice and useful trick to have the same dataset per every run and also creating new data that won't mess up with the previous ones due to object references and how the data is loaded in memory.

6

In Table (1), the accuracy along the epochs is shown.
In Figure (1), I provide the plot with epochs on the x-axis where two graphs are plotted: the train and validation accuracies. The context is the one from the previous Table (1).

7

Overall, after assessing the performances of the network on the unseen test set the model is realiable in estimating the classification of the images with a best accuracy slightly lower XXX than the one on the validation set, and the average of the accuracies are around XXX, with a variance of XXX.
The delusional result is due to the nature of Feed Forward Neural Network: the learning is focused on pixel per pixel, while a desired and more useful approach would be the

Table 1: Training and Validation Accuracy per Epoch for FFNN

Epoch	Train Accuracy (%)	Validation Accuracy (%)
1	19.87	18.41
11	47.88	31.77
21	72.99	38.99
31	85.37	31.77
41	98.92	34.66
51	77.51	25.63
61	99.91	33.21
71	100.00	32.13
81	100.00	31.41
91	100.00	32.49
101	100.00	31.77
Best validation accuracy: 38.99%		

Figure 1: Train & validation accuracies for FFNN

ones resembling human interaction, i.e. a broad overview of the whole image, analyzing it at chunks.
As it is difficult for us to spot an image from a puzzle piece, the FFNN has a metaphorically resembling trouble.
For us humans, if we might be able to immediately recognize a person (macro-concept) from its smile (micro-concept). A similar concept is the key to improve the results and lead to the CNN model approach in Task 2.

8 BONUS

The bonus was completed using the mean and average for each RGB channel. In Figure (2) and Table (2), the results thought show a similar slightly lower accuracy value during training.

Table 2: Training and Validation Accuracy per Epoch for FFNN with features

Epoch	Train Accuracy (%)	Validation Accuracy (%)
1	17.89	24.55
11	31.35	25.27
21	31.44	28.16
31	31.44	29.96
41	31.98	27.44
51	32.16	32.13
61	32.43	32.13
71	32.88	32.85
81	34.33	29.96
91	34.15	31.05
101	34.15	33.94
Best validation accuracy: 34.30%		

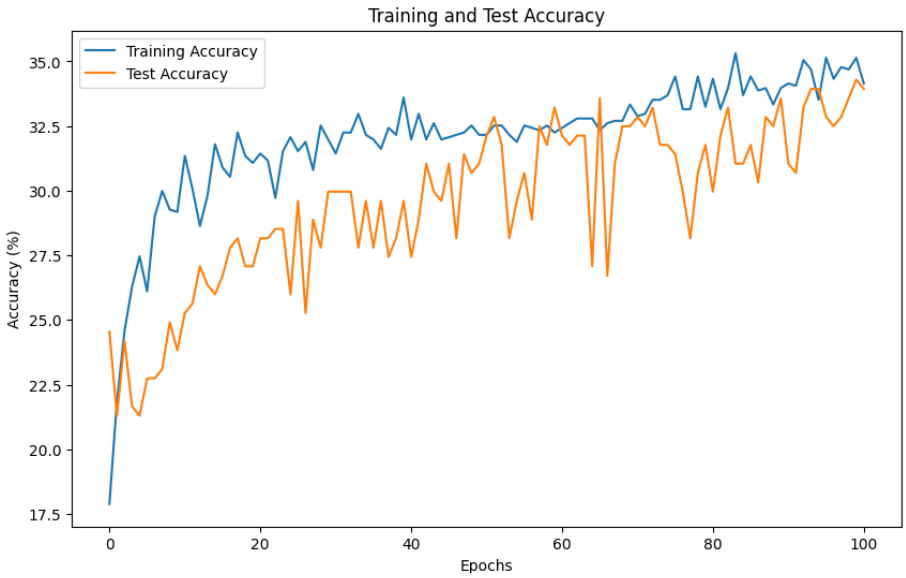

Figure 2: Train & validation accuracies for FFNN with features

The validation results highlighting XXX with variance of XXX. I would have like to put more features in order to get a better model.
The statistical comparison with the FFNN model without features yield the following results: there XXXXX statistical significantly difference with the $p - value = 0.05$. The best result is yild by model XXXXX with best accuracy XXXX and variance of XXXX.
It suffers from the same disadvantages from FFNN discussed previously, so the improvement is sadly bounded by the model nature.

2 Image Classification with Convolutional Neural Networks (CNN)

Implement a multi-class classifier (CNN model) to identify the class of the images: 'CHEETAH' ,'OCELOT', 'SNOW LEOPARD', 'CARACAL', 'LIONS', 'PUMA', 'TIGER'.

1. Follow steps 1 and 2 from T1 to prepare the data.
2. Build a CNN of your choice, following these constraints:
   - use 3 convolutional layers.
   - use 3 pooling layers.
   - use 3 dense layers (output layer included).
3. Train and validate your model. Choose the right optimizer and loss function.
4. Follow steps 5 and 6 of T1 to assess performance.
5. Qualitatively and **statistically** compare the results obtained in T1 with the ones obtained in T2. Explain what you think the motivations for the difference in performance may be.
6. **Bonus** (Optional) Tune the model hyper-parameters with a **grid search** to improve the performances (if feasible).
   - Perform a grid search on the chosen ranges based on hold-out cross-validation in the training set and identify the most promising hyper-parameter setup.
   - Compare the accuracy on the test set achieved by the most promising configuration with that of the model obtained in point 4. Are the accuracy levels **statistically** different?

3

The following train parameters were chosen for the model. The Optimizer chosen is XXXX because XXXX. The loss function used is the CrossEntropyLoss function, which fits the categorical nature of the classification problem.

4

In Table (3), the accuracy along the epochs is shown.
In Figure (3), I provide the plot with epochs on the x-axis where two graphs are plotted: the train and validation accuracies. The context is the one from the previous Table (3).
Overall, after assessing the performances of the network on the unseen test set the model is realiable in estimating the classification of the images with a best accuracy slightly lower XXX than the one on the validation set, and the average of the accuracies are around XXX, with a variance of XXX.

5

The statistical comparison between the simple FFNN and the current CNN model yield the following results: there XXXXX statistical significantly difference with the $p - value = 0.05$. The best result is yield by model XXXXX with best accuracy XXXX and variance of XXXX.

Table 3: Training and Validation Accuracy per Epoch for CNN

Epoch	Train Accuracy (%)	Validation Accuracy (%)
1	18.97	19.13
11	95.66	49.46
21	100.00	53.79
31	100.00	54.51
41	100.00	54.51
51	100.00	54.51
61	100.00	54.87
71	100.00	55.23
81	100.00	54.87
91	100.00	54.87
101	100.00	54.87
Best validation accuracy: 55.23%		

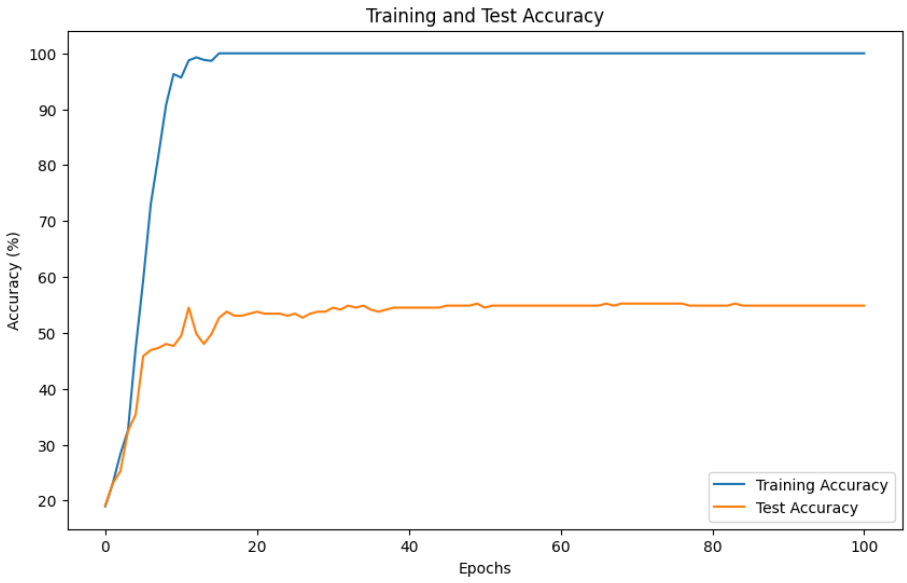

Figure 3: Train & validation accuracies for CNN

6 (present only in README.md)

Apply image manipulation and augmentation techniques in order to improve the performance of your models. Evaluate the performance of the model using the new images and compare the results with the previous evaluation performed in part 3. Provide your observations and insights.
In Table (4), the accuracy along the epochs is shown.
In Figure (4), I provide the plot with epochs on the x-axis where two graphs are plotted: the train and validation accuracies. The context is the one from the previous Table (4).
Overall, after assessing the performances of the network on the unseen test set the model is realiable in estimating the classification of the images with a best accuracy slightly lower XXX than the one on the validation set, and the average of the accuracies are around XXX, with a variance of XXX.

Table 4: Training and Validation Accuracy per Epoch for Augmented CNN

Epoch	Train Accuracy (%)	Validation Accuracy (%)
1	17.43	18.77
11	67.03	67.15
21	82.84	79.78
31	88.89	77.26
41	94.40	77.62
51	95.03	80.14
61	95.93	79.78
71	95.48	82.31
81	97.47	85.20
91	97.29	84.84
101	97.83	82.67
Best validation accuracy: 85.56%		

Figure 4: Train & Validation accuracies for Augmented CNN

7 BONUS

For the grid search I choose to change the following hyperparameters:

1 Learning rate: [0.001, 0.01, 0.1]

2 Batch size: [32, 64, 128]

The epochs were fixed at 100 as defined in configuration.
The results of such grid search yield me a model with peak validation accuracy of 62.09% with the following best parameters:

- batch size = 64
- learning rate = 0.001
- epochs = 100

In Table (5), the accuracy along the epochs is shown.

Table 5: Training and Validation Accuracy per Epoch for CNN with Grid Search

Epoch	**Train Accuracy (%)**	**Validation Accuracy (%)**
1	18.16	17.33
11	99.46	58.48
21	99.64	55.96
31	100.00	61.73
41	100.00	61.37
51	100.00	61.01
61	100.00	60.29
71	100.00	59.93
81	100.00	59.57
91	100.00	59.21
101	100.00	58.48
Best validation accuracy: 62.09%		

3 Transfer Learning

This task involves loading the VGG19 model from PyTorch, applying transfer learning, and experimenting with different model cuts. The VGG19 architecture have 19 layers grouped into 5 blocks, comprising 16 convolutional layers followed by 3 fully-connected layers. Its success in achieving strong performance on various image classification benchmarks makes it a well-known model.
Your task is to apply transfer learning with a pre-trained VGG19 model. A code snippet that loads the VGG19 model from PyTorch is provided. You'll be responsible for completing the remaining code sections (marked as TODO). Specifically:

1. The provided code snippet sets param.requires_grad = False for the pre-trained VGG19 model's parameters. Can you explain the purpose of this step in the context of transfer learning and fine-tuning? Will the weights of the pre-trained VGG19 model be updated during transfer learning training?

2. We want to transfer learning with a pre-trained VGG19 model for our specific classification task. The code has sections for __init__ and forward functions but needs to be completed to incorporate two different "cuts" from the VGG19 architecture. After each cut, additional linear layers are needed for classification (similar to Block 6 of VGG19). Implement the __init__ and forward functions to accommodate these two cuts:

   - This cut should take the pre-trained layers up to and including the 11th convolution layer (Block 4).
   - Cut 2: This cut should use all the convolutional layers from the pre-trained VGG19 model (up to Block 5).

     Note after each cut take the activation function and the pooling layer associated with the convolution layer on the cut

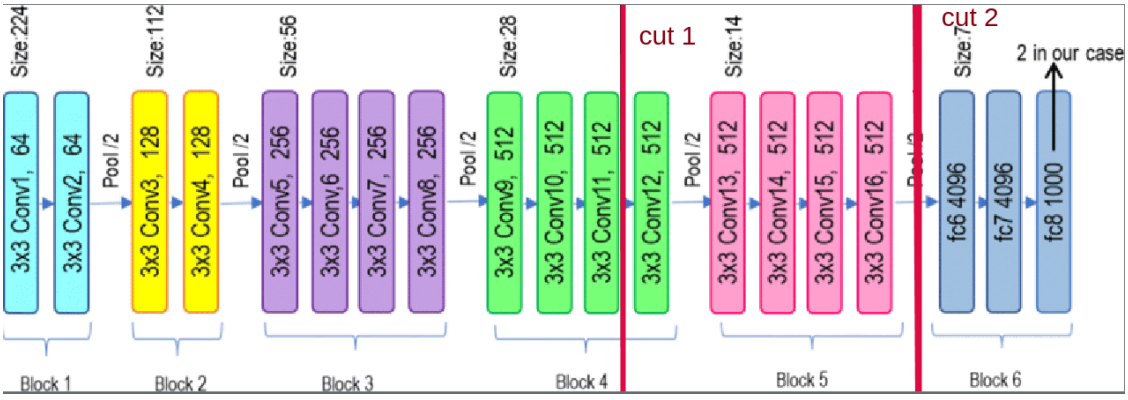

Figure 5: Cuts in VGG19

3. In both cases, after the cut, add a sequence of layers (of your choice) with appropriate activation functions, leading to a final output layer with the desired number of neurons for your classification task. Train the two models (one with Cut 1 and another with Cut 2) on your chosen dataset. Once training is complete, compare their performance statistically.

4. Based on the performance comparison, discuss any observed differences between the two models. What could be the potential reasons behind these results?

5. BONUS (optional): Try different cuts in each block of VGG19, and plot one single figure with all the train-validation-test accuracies. Explain in detail the reasons behind the variation of results you get.

1

Within the context of transfer learning, the model VGG19 from PyTorch is loaded with the architecture and weights of a pre-trained model. The purpose of setting `param.requires_grad =False` is to **freeze the weights** of the model, so that the model does not update the weights during the training of the new model. In fact, it would not make sense to update the weights of the pre-trained model in the context of transfer learning. Particularly, transfer learning is a tecnique where a model for one task, in this case the VGG19 for image classification, is reused as the starting point for a model on a second task, where the fine tuning to train the model over our feline dataset happens. Hence, during transfer learning traning, the weights of the pre-trained VGG19 will XXX be updated.

2

To select the first cut, we need to from the firt feature children of the VGG19 architecture to index 25, in order to get the 11th convolution layer and its activation function and pooling layers associated. The correctness can be easily checked thanks to the already templated `print(self.features)` after the cut, so you will see all the requirements satisfied.
For the second cut, we trivially load all the features from the model.

3

For fine tuning we put also a sequence of 3 linear layers with relu activation function complementing with a dropout strategy to try to avoid overfitting. Starting from flattened image size on the VGG19, for which I wrote a dummy input in the constructor to retrieve what would be the flatten image after the features of the pre-trained model up to its 11th convolution layer and its associated activation functions and pools for cut1. The number of neuron in the added sequence of layers were 256 and 128, ending with an output correspoding to our cat types in the dataset, i.e. 6.
In Table (6), the accuracy along the epochs is shown.
In Figure (6), I provide the plot with epochs on the x-axis where two graphs are plotted: the train and validation accuracies. The context is the one from the previous Table (6).
Overall, after assessing the performances of the network on the unseen test set the model is realiable in estimating the classification of the images with a best accuracy slightly lower XXX than the one on the validation set, and the average of the accuracies are around XXX, with a variance of XXX.
In Table (7), the accuracy along the epochs is shown.
In Figure (7), I provide the plot with epochs on the x-axis where two graphs are plotted: the train and validation accuracies. The context is the one from the previous Table (7).
Overall, after assessing the performances of the network on the unseen test set the model is realiable in estimating the classification of the images with a best accuracy slightly lower XXX than the one on the validation set, and the average of the accuracies are around XXX, with a variance of XXX.
The statistical comparison between the cut1 and the cut2 pre-trained and fine-tuned models yield the following results: there XXXXX statistical significantly difference with

Table 6: Training and Validation Accuracy per Epoch for fine-tuned VGG19 with Cut 1

Epoch	Train Accuracy (%)	Validation Accuracy (%)
1	31.89	58.48
11	71.09	84.48
21	77.69	88.09
31	80.94	85.56
41	81.66	87.00
51	82.57	85.92
61	83.29	85.92
71	85.00	90.61
81	78.41	86.28
91	82.57	84.12
101	85.73	86.28
Best validation accuracy: 90.61%		

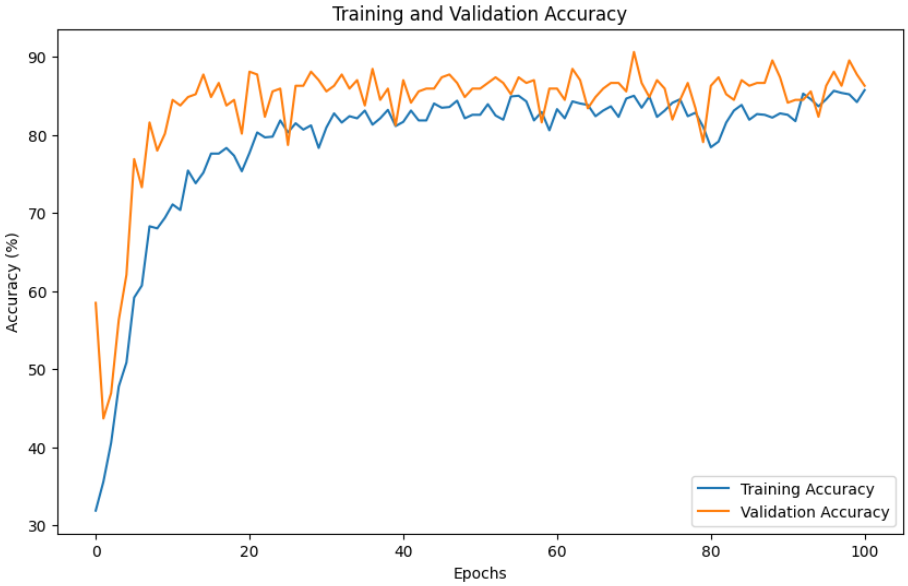

Figure 6: Train & validation accuracies for fine-tuned VGG19 with Cut 1

Table 7: Training and Validation Accuracy per Epoch for fine-tuned VGG19 with Cut 2

Epoch	Train Accuracy (%)	Validation Accuracy (%)
1	61.07	92.78
11	99.46	95.31
21	98.55	95.31
31	99.28	94.22
41	98.92	95.67
51	99.37	93.86
61	99.64	93.86
71	99.64	94.58
81	99.10	92.06
91	99.46	94.22
101	99.91	94.22
Best test accuracy: 96.75%		

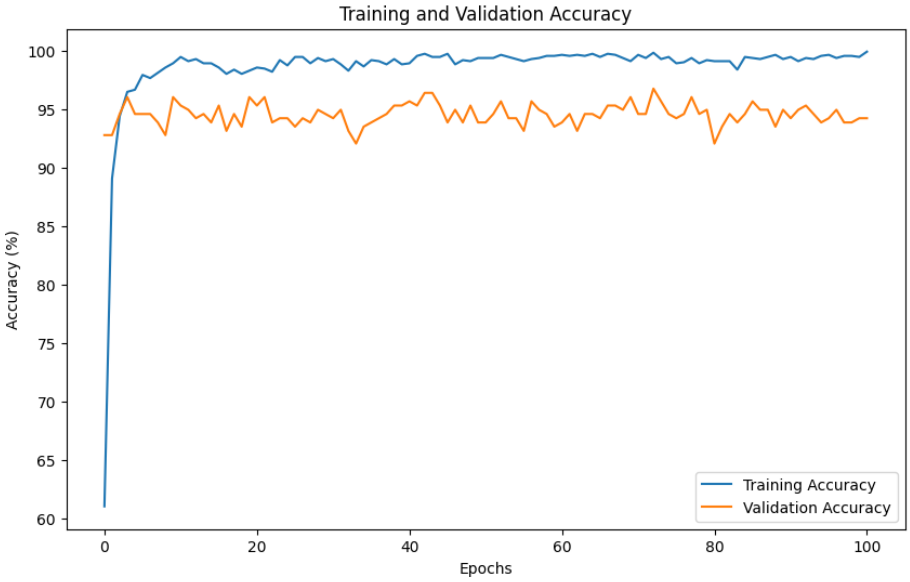

Figure 7: Train & validation accuracies for fine-tuned VGG19 with Cut 2

the $p - value = 0.05$. The best result is yield by model XXXXX with best accuracy XXXX and variance of XXXX.

4

Based on the performance comparison, the reason why the cut 2 performs significantly better than cut 1 lies within the transfer learning of the VGG19 itself: its architecture is meant to be used XXXX.
Hence, the potential reason of the lower accuracy for cut 1 woud be indeed in the difference of XXX.

5 BONUS

For the bonus part, I chose the following different cuts among the blocks of the VGG19 model:

- **Cut A** only first block
- **Cut B** 2 blocks
- **Cut C** 3 blocks
- **Cut D** 4 blocks
- **Cut E** 5 blocks

In Figures (9, 8), the result plot of the train-test-validation of accuracies are presented in two flavours respectively: an histogram and a line plot.

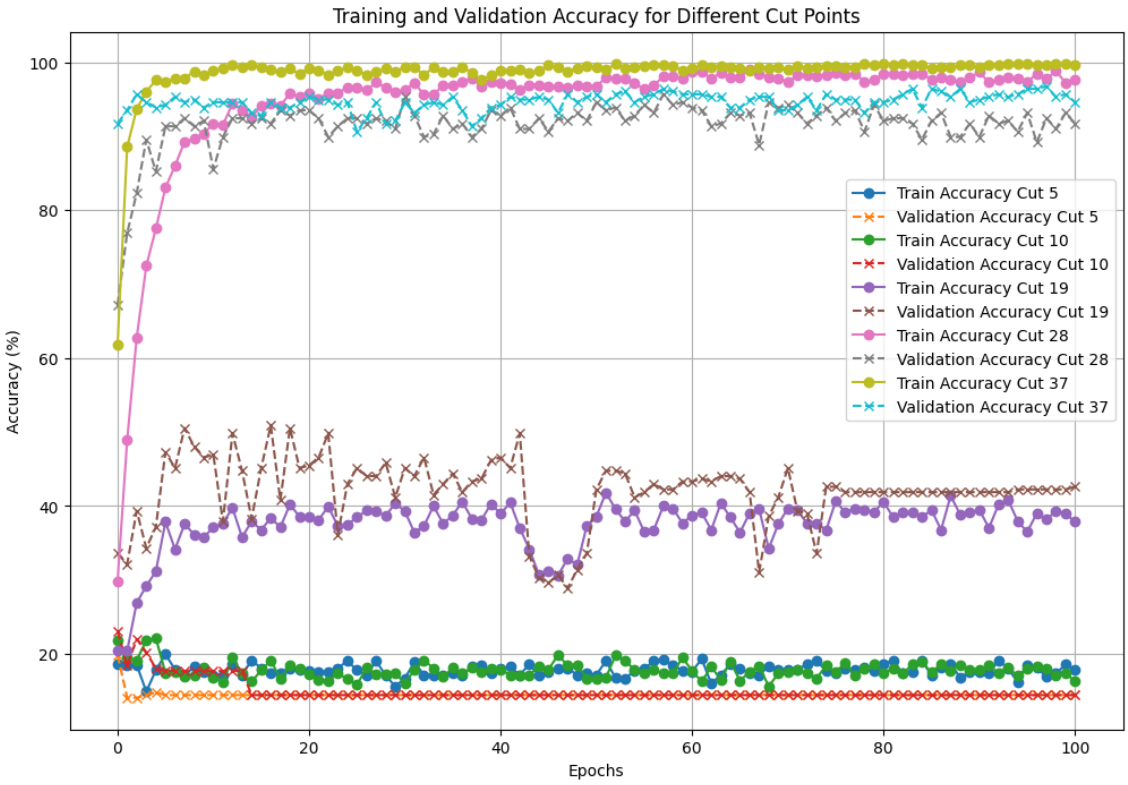

Figure 8: train-test-validation accuracies plot

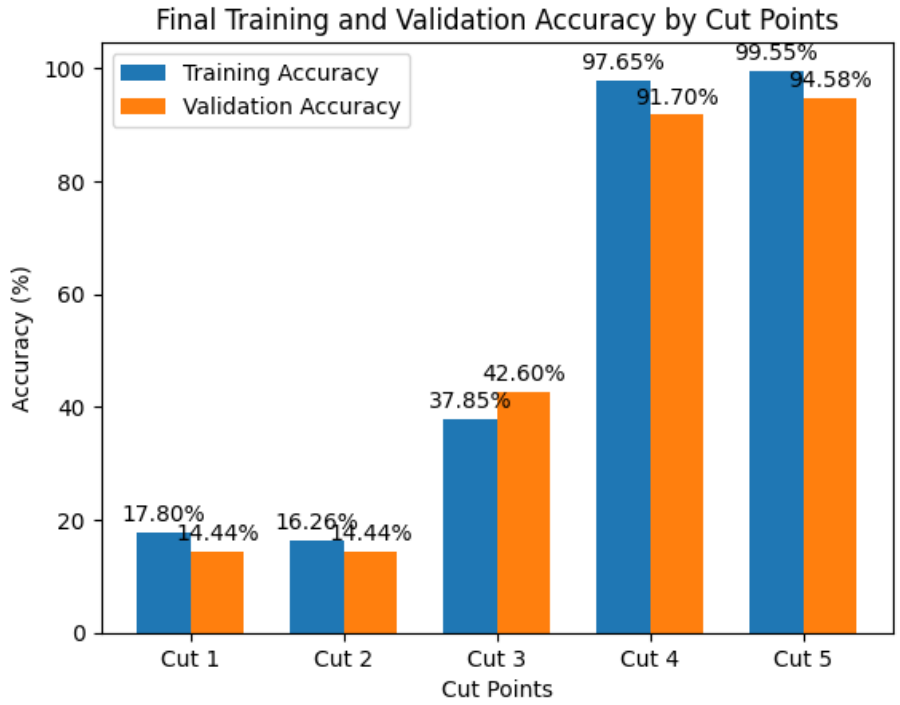

Figure 9: train-test-validation accuracies histogram

The reason behind such variation of results leads to a stronger support of the previously mentioned difference between cut1 and cut2: XXXXXX.
Time increases with layers, it makes sense to choose fewer sufficient layers.
In Tables (**??**, **??**, **??**, **??**, **??**,) showcase the accuracies respectively for Cut A, B, C, D, E.

Table 8: Training and Validation Accuracy per Epoch for Model Cut A

Epoch	**Train Accuracy (%)**	**Validation Accuracy (%)**
1	18.61	19.86
11	17.16	14.44
21	17.71	14.44
31	16.71	14.44
41	18.07	14.44
51	17.16	14.44
61	17.62	14.44
71	17.89	14.44
81	18.61	14.44
91	17.62	14.44
101	17.80	14.44
Best validation accuracy: 19.86%		

Table 9: Training and Validation Accuracy per Epoch for Model Cut B

Epoch	Train Accuracy (%)	Validation Accuracy (%)
1	21.86	23.10
11	16.80	17.69
21	17.25	14.44
31	16.08	14.44
41	17.89	14.44
51	16.71	14.44
61	17.71	14.44
71	17.52	14.44
81	17.43	14.44
91	17.80	14.44
101	16.26	14.44
Best validation accuracy: 23.10%		

Table 10: Training and Validation Accuracy per Epoch for Model Cut C

Epoch	Train Accuracy (%)	Validation Accuracy (%)
1	20.42	33.57
11	37.13	46.93
21	38.57	45.49
31	39.30	45.13
41	38.93	46.57
51	38.57	42.24
61	38.75	43.32
71	39.66	45.13
81	40.56	41.88
91	39.48	41.88
101	37.85	42.60
Best validation accuracy: 50.90%		

Table 11: Training and Validation Accuracy per Epoch for Model Cut D

Epoch	Train Accuracy (%)	Validation Accuracy (%)
1	29.81	67.15
11	91.69	85.56
21	95.84	93.50
31	96.30	95.31
41	97.20	92.78
51	96.75	94.58
61	98.64	93.86
71	97.29	94.22
81	98.46	92.06
91	98.64	89.89
101	97.65	91.70
Best validation accuracy: 95.67%		

Table 12: Training and Validation Accuracy per Epoch for Model Cut E

Epoch	**Train Accuracy (%)**	**Validation Accuracy (%)**
1	61.79	91.70
11	98.92	94.58
21	99.19	95.31
31	99.37	94.58
41	98.83	94.22
51	99.37	95.67
61	99.19	95.67
71	99.01	93.50
81	99.82	94.58
91	99.28	94.95
101	99.55	94.58
Best validation accuracy: 96.75%		