Machine Learning
Università della Svizzera italiana

Assignment 2

Morales 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. Errata Corrige: The dataset provided has no 'SNOW LEOPARD' class.
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** 32
- **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.
Reproducibility: The seed is set initially to `np.random.seed()`, `torch.manual_seed()` and passed to the functions `train_test_split(..., random_state=...)` 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. Proof of the same dataset per every run is the same accuracy and loss values across the Task 1 and its bonus because of determinism in seed and FFNN operations. Despite this, only the first Task 1 and its bonus lead to reproducible results. The reason behind is that despite setting the seeds, the table results in this report can be different because the torch library performs non deterministic operations and algorithm choices for convolutions, used from Task 2 onwards. More on that on CUDA convolution determinism - PyTorch documentation .

Reproduction Package

The models used for the report are stored in *.pth* PyTorch format in my personal USI OneDrive shared folder . All the plots and tables are generated using such models and the same dataset thanks to the seed set in the notebook. All the code has been developed in a GitHub repository .

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).

Table 1: Training and Validation Accuracy per Epoch for FFNN

Epoch	Train Accuracy (%)	Validation Accuracy (%)
1	19.87	25.36
11	47.88	28.99
21	72.99	39.86
31	85.37	34.78
41	98.92	38.41
51	77.51	32.61
61	99.91	37.68
71	100.00	33.33
81	100.00	34.06
91	100.00	34.06
101	100.00	33.33
Best validation accuracy: 39.86%		

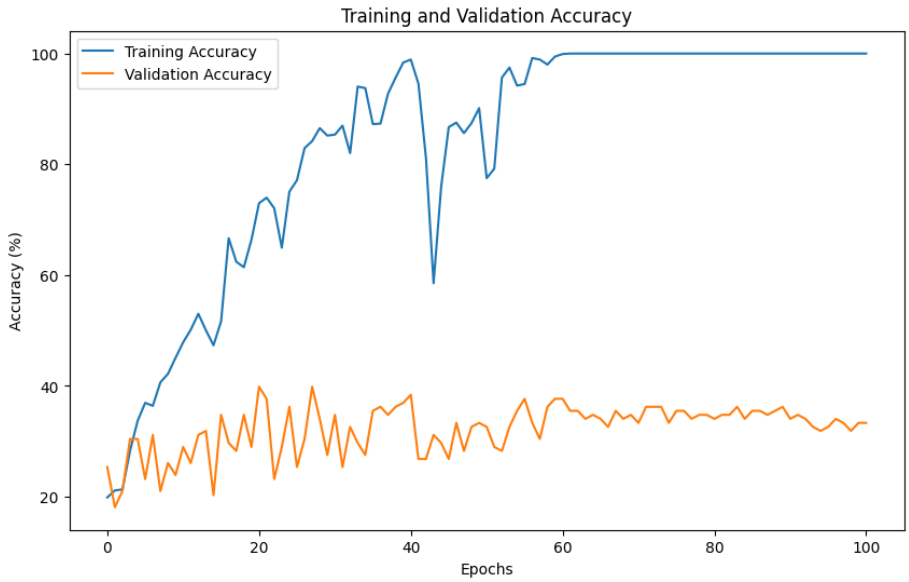

Figure 1: Train & validation accuracies for FFNN

7

Overall, after assessing the performances of the network on the unseen test set, the model is not realiable in estimating the classification of the images: its test accuracy is 38.13%, slightly lower than the best one of the validation set, meaning very poor performance thought is already above the random guess of $1/6 \approx 16.67\%$. The variance of the accuracies is unrealiably big: 209.89. To complement, the Loss averaged is 1.90.
The delusional and unreliable 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 ones resembling human interaction, i.e. a broad overview of the whole image, analyzing it in chunks.
As it is difficult for us to spot an image from a puzzle piece, the FFNN has a metaphorically resembling trouble with image pixels.
For us humans, we might be able to 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 with the following features: the mean and average for each RGB channel.
In Figure (2) and Table (2), the results 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	19.15	20.29
11	30.80	21.01
21	31.89	21.74
31	32.34	25.36
41	31.71	29.71
51	33.60	23.91
61	33.51	31.16
71	34.24	26.81
81	34.06	29.71
91	34.96	29.71
101	35.41	26.09
Best validation accuracy: 32.61%		

The performance assessment done on the unseen same test set yields an accuracy of: 35.25%. It highlights a slightly lower accuracy with respect to the model from T1 with variance of 36.85, which is far better than the previous model, but still not enough to be considered reliable for image classification. To complement, the Loss averaged is 1.58.
Despite more features regarding the image could have been added, one should be careful to reason about the logical implication of it. As previously said, the Feed Forward Neural Network is not the best model for image classification due to its nature of pixel per pixel analysis. Most of these features are not useful for the model to learn from: althought same type of lions might share the same color of the fur, most of the proposed felines share a common color palette.

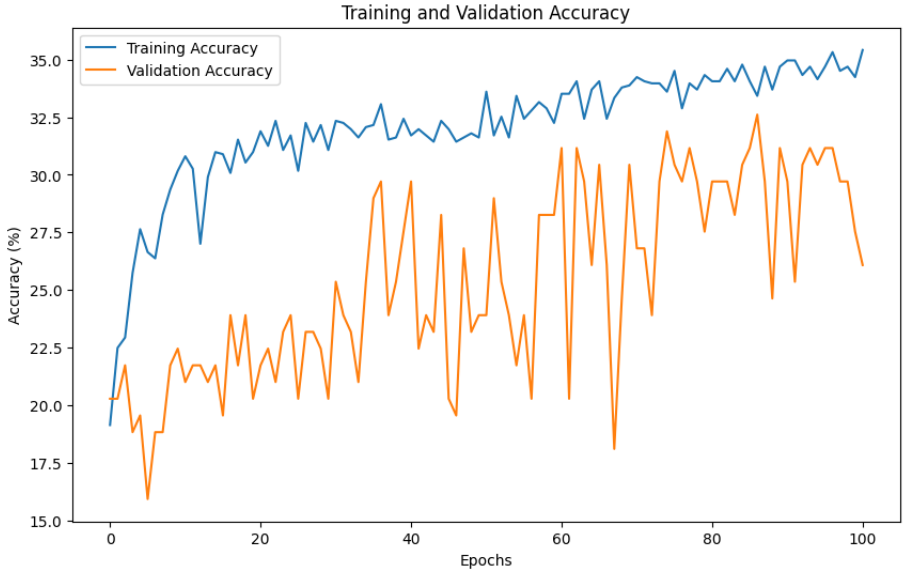

Figure 2: Train & validation accuracies for FFNN with features

Moreover, the statistical comparison with p-value test set to $p = 0.05$ between the FFNN model without features yield the following results: there is **no statistical significantly difference**. The resulted values from statistical comparison are: $t\_statistic = -0.15, p\_value = 0.89$.
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'. Errata Corrige: The dataset provided has no 'SNOW LEOPARD' class.

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. Errata Corrige: from 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.
7. **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: for the optimizer, the **Adam optimizer** was chosen because of its adaptative learning rate and its ability to handle sparse gradients on noisy problems. Instead, for the loss function, the **CrossEntropyLoss** was chosen because it is suitable for multi-class classification problems, perfectly fitting its categorical nature.

4

In Table (3), the validation 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 same unseen test set, the model is estimating the classification of the images with an accuracy of 65.47%, slightly lower than the best one on the validation set. The variance of the accuracy is 34.85,

and to complement, the loss averaged is 2.2034. The **overfitting** phenomenon is clearly present after the 11th epoch, where the validation accuracy starts to decrease while the training accuracy peaks to 100. The model performs better than the odds of a coin flip!

Table 3: Training and Validation Accuracy per Epoch for CNN

Epoch	Train Accuracy (%)	Validation Accuracy (%)
1	19.15	13.04
11	98.92	61.59
21	100.00	68.12
31	100.00	67.39
41	100.00	67.39
51	100.00	67.39
61	100.00	67.39
71	100.00	67.39
81	100.00	67.39
91	100.00	67.39
101	100.00	67.39
Best validation accuracy: 68.12%		

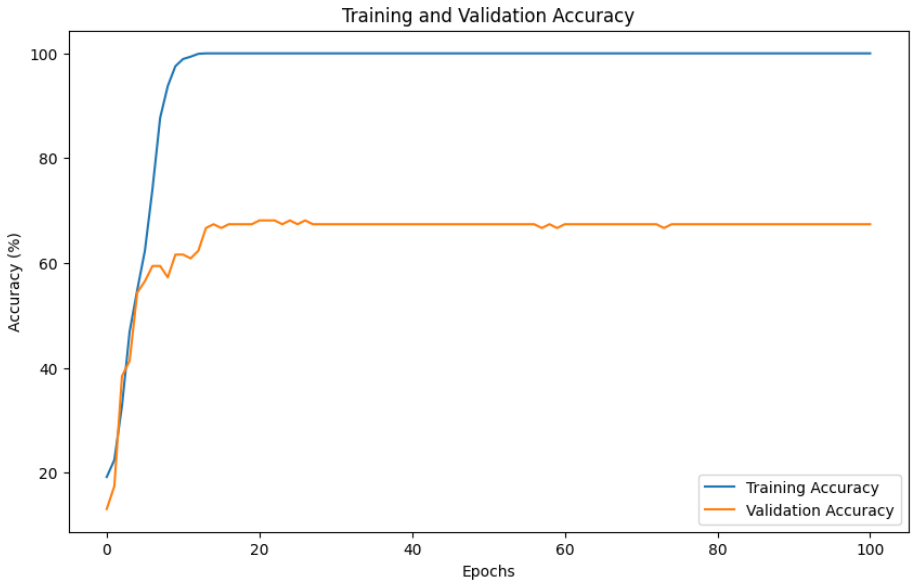

Figure 3: Train & validation accuracies for CNN

5

The statistical comparison with p-value test set to $p = 0.05$, between the simple FFNN from Task 1 and the current CNN model yield the following results: **there is statistical significantly difference**. The resulted values from statistical comparison are: $t\_statistic = -3.78, p\_value = 0.019$.
The accuracy more than doubled from the FFNN model to the CNN model. The reason for such improvement is the nature of the CNN model: it is able to capture the spatial dependencies in the image through the application of relevant filters. The CNN model is able to learn the features of the image in a more efficient way, than pixel per pixel

analysis of the FFNN model, and it is able to generalize better to unseen data using the locality principle: the closer the pixels, the more they are related to each other.
Now, instead that a puzzle piece, imagine that you are able to see a cropped square of part of the image with a blurring effect. Well, patterns are more easily recognizable and the convolutional layers are able to learn these patterns. Image recognition is a task that suitable for CNN models.

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 same unseen test set the model is more realiable in estimating the classification of the images with a test accuracy of 85.61%, slightly lower than the best one from validation. Thought, its variance is a bit high: 50.78, leading to skepticality on new unseed data not from the current dataset. To complement, the test loss averaged is 0.65.

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

Epoch	**Train Accuracy (%)**	**Validation Accuracy (%)**
1	17.98	14.49
11	67.39	56.52
21	83.92	81.88
31	90.70	72.46
41	93.32	81.16
51	94.49	81.16
61	97.02	82.61
71	98.01	85.51
81	98.46	86.23
91	97.20	84.78
101	98.19	78.99
Best validation accuracy: 87.68%		

The statistical comparison with p-value test set to $p = 0.05$, between the simple CNN model from Task 1 and the current augmented CNN model yield the following results: **there is statistical significantly difference**. The resulted values from statistical comparison are: $t\_statistic = -5.05, p\_value < 0.01$.
The accuracy gained 20% with respect to the simple CNN model. The reason for such improvement is the augmentation provided transforming the input images throughtout the CNN model: techniques as increasing/decreasing the brightness, contrast and saturation of the images can potentially highlight the features of the images characterizing the feline types. Cropping could also lend a hand in the model, zooming in the most important parts of the image in order for the principle of locality to work better. Other random transformations as horizontal flips and random rotations doubtably help the model to generalize better to unseen data, in my opinion.

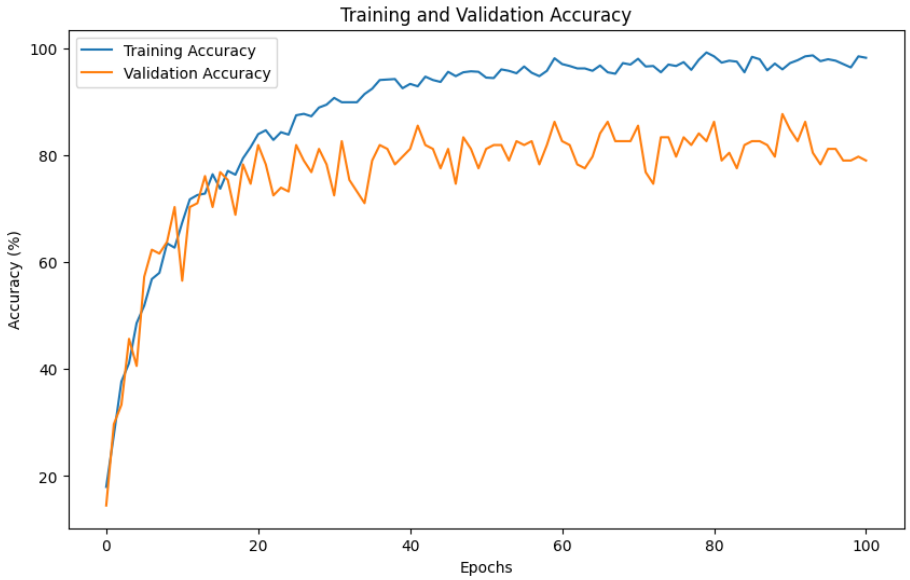

Figure 4: Train & Validation accuracies for Augmented CNN

The result is a model that is able to generalize better to unseen data, yielding satisfying results on the test set. The main advantages here are grounded up from the CNN properties with the proposed augmentation techniques in order to apply a bit of fine tuning to the already working model.

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 aim of this grid search is to find the best hyperparameters for the model with respect to the learning rate and the batch size, in order to improve the performances of the model. The results of such grid search yield me a model with peak validation accuracy of 60.43% with the following best parameters:

- batch size = 32
- learning rate = 0.001

In Table (5), the accuracy along the epochs is shown. The data shows an **overfitting** phenomenon after the 21st epoch, where the validation accuracy starts to decrease while the training accuracy peaks to 100. This model has an accuracy of 60.43%, slightly less accuracy the the previous simple CNN model of Task 2 of 65.47%. The variance of the accuracy is 71.49, which is more than the simple CNN model. to complement the loss averaged is 2.65. Even the peak validation accuracy is less than the one from the simple CNN model. By performing a statistical comparison with the same settings as the previous ones, the results yield that there is **no statistical significantly difference** between the simple CNN model from Task 2 and the current grid searched CNN model. In fact, the resulted best statistical parameters are the same configuration choosen to train the simple CNN model from Task 2: the 2 model have equivalent initial hyperparameters!

Table 5: Training and Validation Accuracy per Epoch for best CNN Grid Searched

Epoch	**Train Accuracy (%)**	**Validation Accuracy (%)**
1	15.36	21.01
11	99.64	55.80
21	100.00	64.49
31	100.00	63.77
41	100.00	63.04
51	100.00	62.32
61	100.00	62.32
71	100.00	62.32
81	100.00	62.32
91	100.00	60.87
101	100.00	60.87
Best validation accuracy: 67.39%		

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

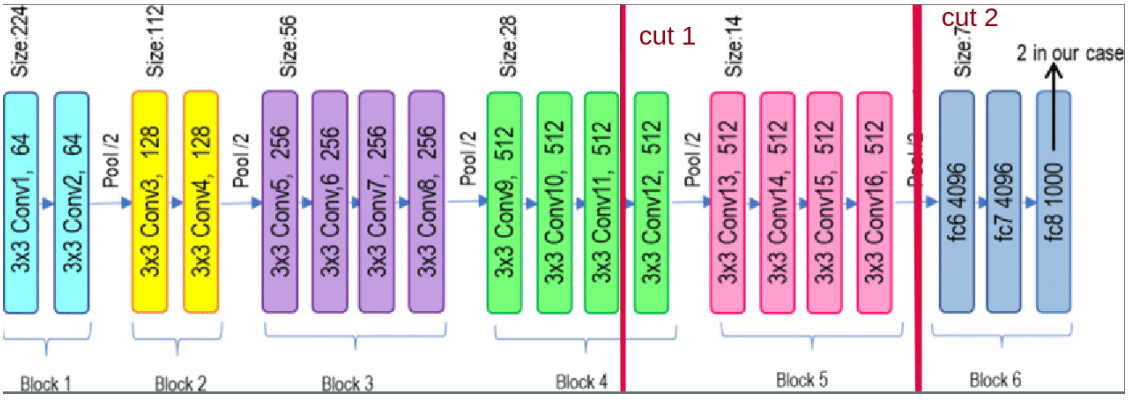

Figure 5: Cuts in VGG19

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 it 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 not 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 a dropout strategy to try to avoid **overfitting**. In the constructor I put a dummy input to retrieve the flatten image size to feed to the first linear layer as input: its purpose is to simulate what happens in the forward function where the features are used on param `x`, then `x` is flattened and passed to the linear layers through our classifiers for fine tuning. The number of neurons 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 (errata corrige put near written dataset specifications for the missing 'SNOW LEOPARD').
In Table (6), the accuracy of Cut 1 along the epochs is shown.
In Figure (6), I provide the plot for Cut 1 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 same unseen test set, the model is realiable in estimating the classification of the images with an accuracy of 87.77%, slightly lower than the best one on the validation set. The variance of the test accuracy is 5.77, with seems to be reasonable and sufficiently reliable for the classification task. To complement, the test loss averaged is 0.35. The odds are better than the ones from the augmented CNN model from Task 2 Bonus!
In Table (7), the accuracy for Cut 2 along the epochs is shown.
In Figure (7), I provide the plot for Cut 2 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 same unseen test set, the model is the most realiable so far in estimating the classification of the images with

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

Epoch	Train Accuracy (%)	Validation Accuracy (%)
1	30.17	65.94
11	68.02	74.64
21	78.41	81.16
31	80.49	84.78
41	85.00	86.23
51	85.09	88.41
61	87.99	86.23
71	86.27	88.41
81	86.90	89.13
91	85.55	84.06
101	85.91	81.88
Best validation accuracy: 90.58%		

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

a test accuracy of 95.68%. It is slightly lower than the best one on the validation set. The variance of the test accuracy is 13.28, which is a bit higher than Cut 1 althought the latter has less accuracy. To complement, the test loss averaged is 0.11.

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

Epoch	Train Accuracy (%)	Validation Accuracy (%)
1	61.07	91.30
11	98.19	94.20
21	99.28	94.93
31	98.83	95.65
41	98.74	95.65
51	99.01	94.20
61	99.28	93.48
71	99.46	94.93
81	99.64	93.48
91	99.73	93.48
101	99.73	92.75
Best validation accuracy: 96.38%		

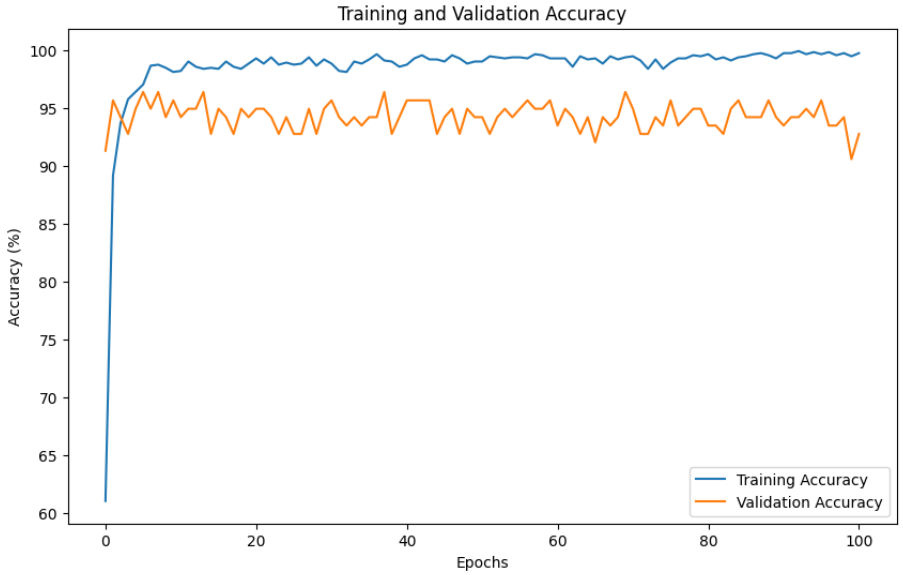

Figure 7: Train & validation accuracies for fine-tuned VGG19 with Cut 2

The statistical comparison with p-value test set to $p = 0.05$, between Cut 1 and Cut 2 models yields the following results: **there is statistical significantly difference**. The resulted values from statistical comparison are: $t\_statistic = -3.82, p\_value < 0.02$.

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: by using the whole architecture of the VGG19 model, meaning all the convolutional layers, the model is able to learn more features from the images and generalize better to unseen data. The cut 1, instead, uses only the first 11 convolutional layers, meaning that the model is not able to

learn as many features as the cut 2 model. That is because the architecture of the VGG19 model is designed to learn features from the images in a hierarchical way, and the more layers are used, the more features are learned. The VGG19 model is meant to be seen as a whole rather than in parts, and the cut 2 model is able to take advantage of this property. Hence, the potential reason of the lower accuracy for cut 1 woud be indeed in the difference of blocks of the architecture of the VGG19 model used for transfer learning.
A metaphor that could be used to explain the difference between cut 1 and cut 2 would be the following: imagine you need an arm to grab something, the arm is divided in shoulder, elbow, hand, fingers, knuckles and nails. Now, you inadvertently decide to enhance the arm with a tool, but you don't want to use the whole arm, just a cut of it. If you happen to choose up until the fingers, you problably will be able to grab the tool and use it for grabbing objects. Maybe choosing up until the knuckles would be enough, as the nails are not that impactful to grabbing objects despite they could potentially help with the grip or the precision, e.g. when we need to open something very small, like unleash a node in our shoes. This kind of scenario is seen afterwards in the Task 3 bonus. Most importantly, if you choose only up until the elbow, il would be difficult to grab anything at all, despite the "architecture" of the arm being amazing as the evolution of our species awarded us with.

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.
Please notice that the histogram in Figure (9) for each cut showcase the following: it compares the best validation accuracy with the correspondent train accuracy for "best" epoch during training, as we would do in early stopping. It compares these 2 bars with the test accuracy, from the model evaluation, and it plots confidence intervals for the test accuracy the give the idea of reliableness of the model.
In the line plot of Figure (8), the train-test-validation accuracies are plotted along the epochs. Each cut has its own color and train and validation have different marker styles. The semi transparent lines are the test accuracy for the cut with the corresponding color, in order to see the final performance of the model along the plot.
The reason behind such variation of results leads to a stronger support of the previously mentioned difference between cut1 and cut2: the design of the architecture of the VGG19 model is meant to be seen as a whole, and, generally speaking, the more layers are used, the more features are learned.
Metaphorically, Cut A and B could be seem as the arm up until the elbow: a poor performance from one of the best models for image classification. The, Cut C could be seen as

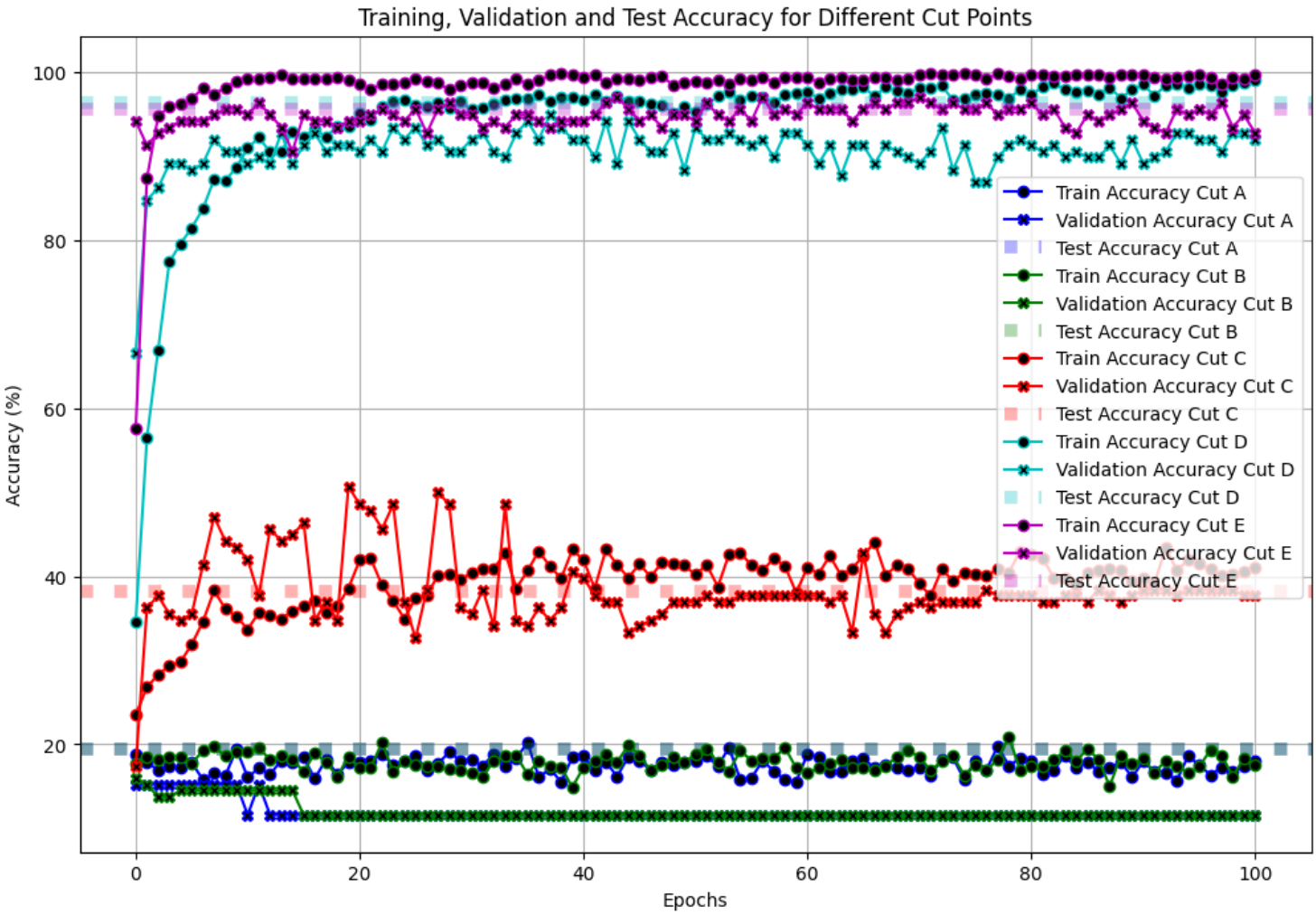

Figure 8: train-validation-test accuracies plot

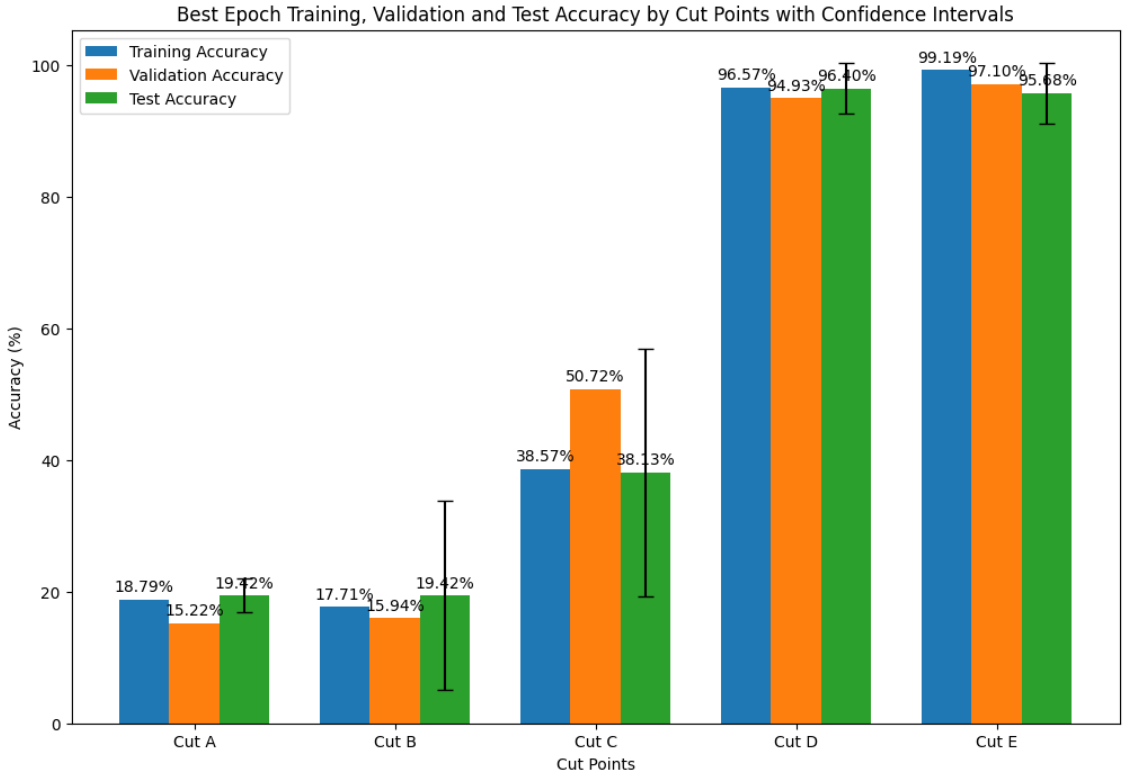

Figure 9: train-test-validation accuracies histogram

the arm up until, but excluding, the hand: could be used but the likehood of success/-efficiency is comparable to less than a coin flip. Lastly, Cut D and E are the equivalent of the whole arm or the arm exluding the nails: reaching the maximum potential of the model, or an accuracy statistically trascurable from the maximum potential.
Then, why not always choose the whole architecture? Well, the more layers are used, the more computational power is needed, and the more time is required to train the model. Fine tuning could be a solution to this problem, but the more layers are used, the more hyperparameters are needed to be tuned. It is a trade-off between computational power, time and hyperparameters tuning.
In Tables (8, 9, 10, 11, 12),the train and validation accuracies along the epochs are shown for respectively Cut A, B, C, D, E.

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

Epoch	Train Accuracy (%)	Validation Accuracy (%)
1	18.79	15.22
11	16.08	11.59
21	17.89	11.59
31	18.16	11.59
41	18.61	11.59
51	18.07	11.59
61	18.88	11.59
71	17.25	11.59
81	17.98	11.59
91	17.98	11.59
101	17.98	11.59
Best validation accuracy: 15.22%		
Test accuracy: 19.42% Test loss averaged: 1.7911 Accuracy variance: 1.7562		

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

Epoch	Train Accuracy (%)	Validation Accuracy (%)
1	17.71	15.94
11	19.06	14.49
21	17.16	11.59
31	16.53	11.59
41	17.25	11.59
51	18.88	11.59
61	16.53	11.59
71	18.52	11.59
81	17.34	11.59
91	18.34	11.59
101	17.52	11.59
Best validation accuracy: 15.94%		
Test accuracy: 19.42% Test loss averaged: 1.7810 Accuracy variance: 53.3510		

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

Epoch	Train Accuracy (%)	Validation Accuracy (%)
1	23.49	17.39
11	33.69	42.03
21	41.92	48.55
31	40.47	35.51
41	41.92	39.86
51	40.20	36.96
61	41.10	37.68
71	39.11	36.96
81	42.64	37.68
91	39.75	38.41
101	41.01	37.68
Best validation accuracy: 50.72%		
Test accuracy: 38.13% Test loss averaged: 1.3020 Accuracy variance: 92.1552		

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

Epoch	Train Accuracy (%)	Validation Accuracy (%)
1	34.60	66.67
11	90.97	89.13
21	95.12	90.58
31	95.57	92.03
41	96.75	92.03
51	95.30	93.48
61	97.65	91.30
71	98.10	89.13
81	97.29	91.30
91	98.55	89.13
101	99.01	92.03
Best validation accuracy: 94.93%		
Test accuracy: 96.40% Test loss averaged: 0.1593 Accuracy variance: 3.9062		

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

Epoch	**Train Accuracy (%)**	**Validation Accuracy (%)**
1	57.54	94.20
11	99.19	94.93
21	98.64	94.20
31	98.74	94.93
41	99.37	94.20
51	98.92	94.20
61	99.37	96.38
71	99.64	97.10
81	99.64	96.38
91	99.73	94.20
101	99.64	92.75
Best validation accuracy: 97.10%		
Test accuracy: 95.68% Test loss averaged: 0.2962 Accuracy variance: 5.4688		