Julien Erbland, Max Henrotin, Cédric Odille

DCS - 233 : Milestone 2

1. Introduction

This project aims to evaluate several machine learning methods using a subset of the Fashion-MNIST dataset. Our primary objective is to recognize different types of clothing articles from images, leveraging the techniques covered in our course. We employ methods such as deep neural networks, including MLP, CNN, and Transformer models for the classification task using PyTorch. Furthermore, we used a PCA for dimensionality reduction. This report details our implementation, results, and analysis, offering insights into the performance and efficiency of each approach.

2. Methods

For this classification task we received as input 60 000 (1,28,28) grayscale images of clothes that we flattened to 784-dimensional vectors. To create a validation set we divided these data and the labels output data in two, with a ratio of 80% for training (xtrain, ytrain) and 20% for the validation set (xtest, ytest). We then decided to normalize the input data in order to prevent extreme values from inadequately influencing the results, however we came to the conclusion that the best results obtained were without normalization. Following this, we gave the option to reduce the dimensionality of our data using the PCA method, this method will be then used to improve the speed of our MLP trainer.

- Principal Component Analysis (PCA):

This method aims to reduce the dimensionality, and thus complexity, of our data by keeping only the dimensions containing the most “importants” signals and least “noises” which is linked to the directions that have a large variance. The PCA method gets this low-dimensional representation of the data by projecting the data on the subspace spanned by the d largest eigenvectors of its covariance matrix.

Following this data processing we trained three models using the Adam optimizer which yielded significantly better results than SGD. We tried differents parameters amongst the three machine learning classifier methods :

- Multilayer Perceptron (MLP):

The MLP is a feedforward artificial neural network which consists of at least three layers of nodes: an input layer, one or more hidden layers, and an output layer. Each node, except for the input node, is a neuron which uses a nonlinear activation function. Each layer is followed by a ReLU activation function. We used the cross entropy loss function as our loss function and the Adam optimizer for training the network.

- Convolutional Neural Network (CNN):

A CNN is a specialized type of artificial neural network designed to process and analyze visual data. The architecture of a CNN is composed of: convolutional layers, which apply convolution operations to the input data to detect local patterns, pooling layers, used to reduce the spatial dimensions of the data, fully connected layers and activation functions.

- Transformer for images:

A transformer is an architecture model using self-attention mechanisms to process data in parallel and capture complex relationships in sequences. We decided to implement a variant of the Vision Transformer (ViT) which processes images by treating them as sequences of patches with positional encoding. It leverages multiple layers of multi-head, self-attention mechanisms to capture complex, long-range dependencies within the image data.

3. Experiment/Results

In the MLP method, we fixed the hidden layer as [128,64] size which gives a great balance between complexity and capacity. To find the best pairs of learning rate and max_iters, we first fixed the learning rate at 1e-5 and plotted the accuracy percentage given different values of max_iters. Our maximum accuracy was 85.1% with lr = 1e-5 (appendix 1). Additionally, we fixed the max_iters at the previously found value to try a different maximum number of learning rates and found that our accuracy was 85.4% (appendix 2). After finding the optimal parameters, we ran a PCA to reduce our computational time (appendix 3). In appendix 4, we looked for the most efficient PCA parameters in terms of trade-off between runtime and accuracy; this corresponds to the setting d = 30 which is equivalent to an explained variance of 74%. However, the optimal threshold for explained variance is 95% which corresponds, in our case, to d= 300 still leading to a gain of 20 seconds (+27% efficiency).

In the CNN method, we can choose different numbers of convolutional layers with different sizes. We decided to fix some parameters to conventional values such as the stride(1), the kernel size for convolutions (3x3) and for pooling (2x2). To find the best learning rate, we fixed max_iters and tried different values. We found that the best lr is either 1e-4 or 1e-3 (appendix 5). We then needed to find the best max_iter. Different epochs do not significantly affect the accuracy but we have a correct accuracy at 20 epoch with a reasonable runtime. The main choice we had was the number of layers and their sizes. We tried different sets of convolutional layers and the best accuracy (89%) was obtained when we had 3 layers (appendix 7). The best is [16,32,64] because it has a lower runtime. We then tested the fully connected layers. It does not impact the accuracy however we can have a better runtime with only one FC layer of size 32 or 64 (appendix 8).

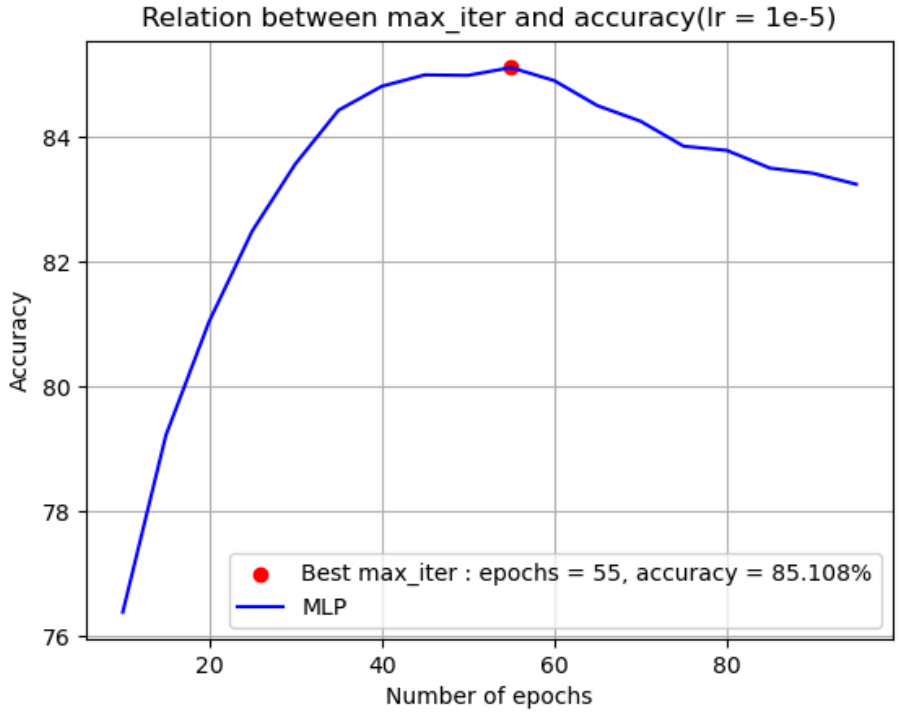
In the Transformer method, we realized that to run one training of our model it took us several hours, therefore it was difficult to fine tune our multiple parameters. Even though, by testing different parameters manually, as described in the appendix 9, we achieved decent performances with n_patch = 14, n_blocks = 4, hidden_d = 64, n_heads = 4 and lr = 1e-3. With these parameters we obtained ~78% accuracy and an F1-score with only 10 epochs.

4. Conclusion

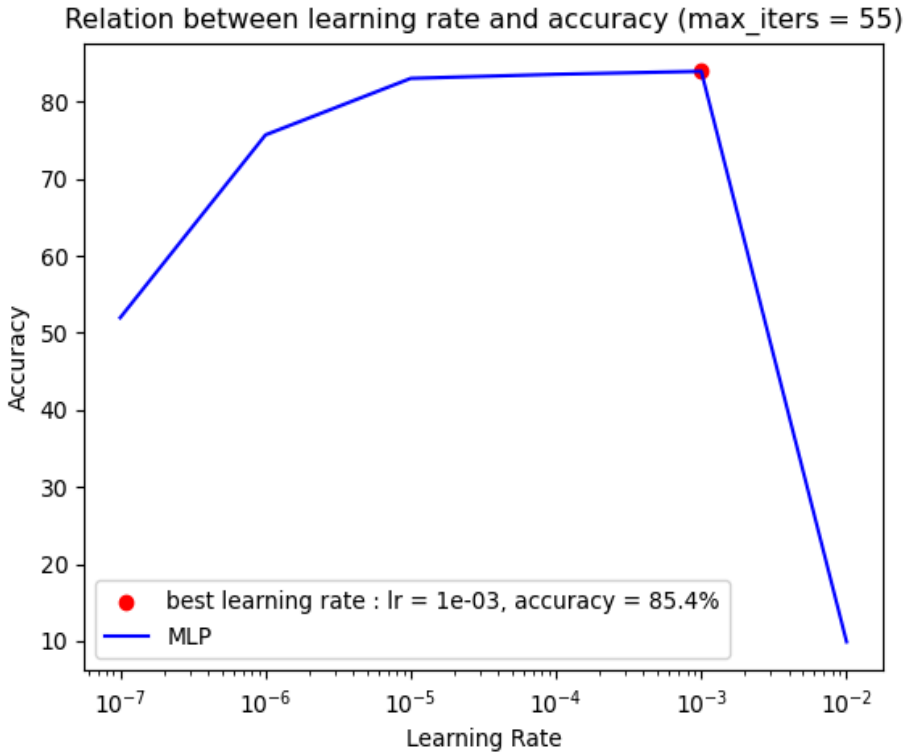
In conclusion, following the training of our three models with various parameters combinations on the fashion MNIST dataset it appears that simple models such as MLP and CNN perform better than the transformer. In addition to the complexity of the multiple layers of multi-head, self-attention, the training time takes 10 times longer to get a performance 10% lower than the 89% accuracy obtained by CNN. However, our significantly fastest model is the combination of MLP and PCA.

Appendix

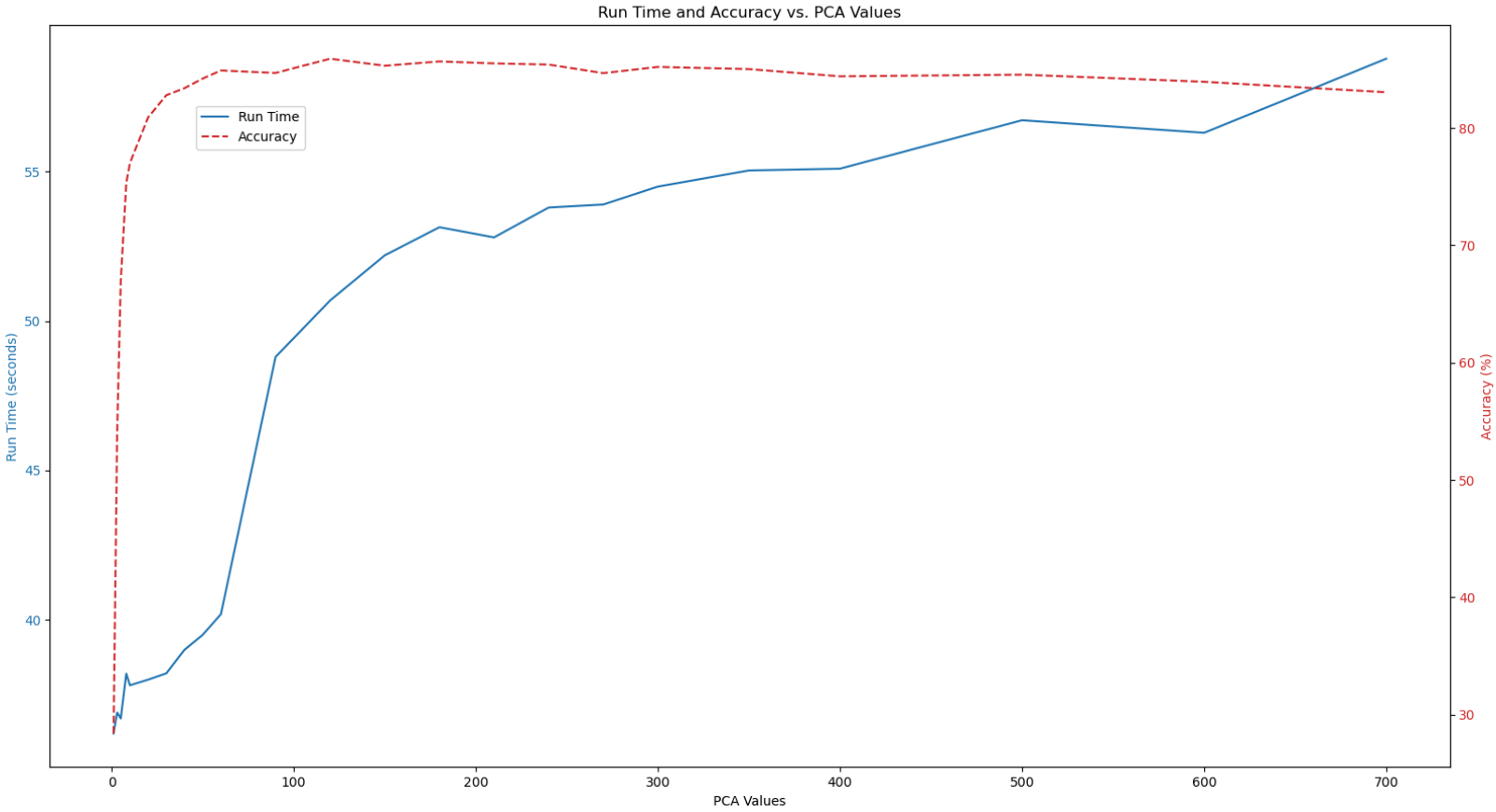
1.Best accuracy for a given max_iter at a fixed learning rate



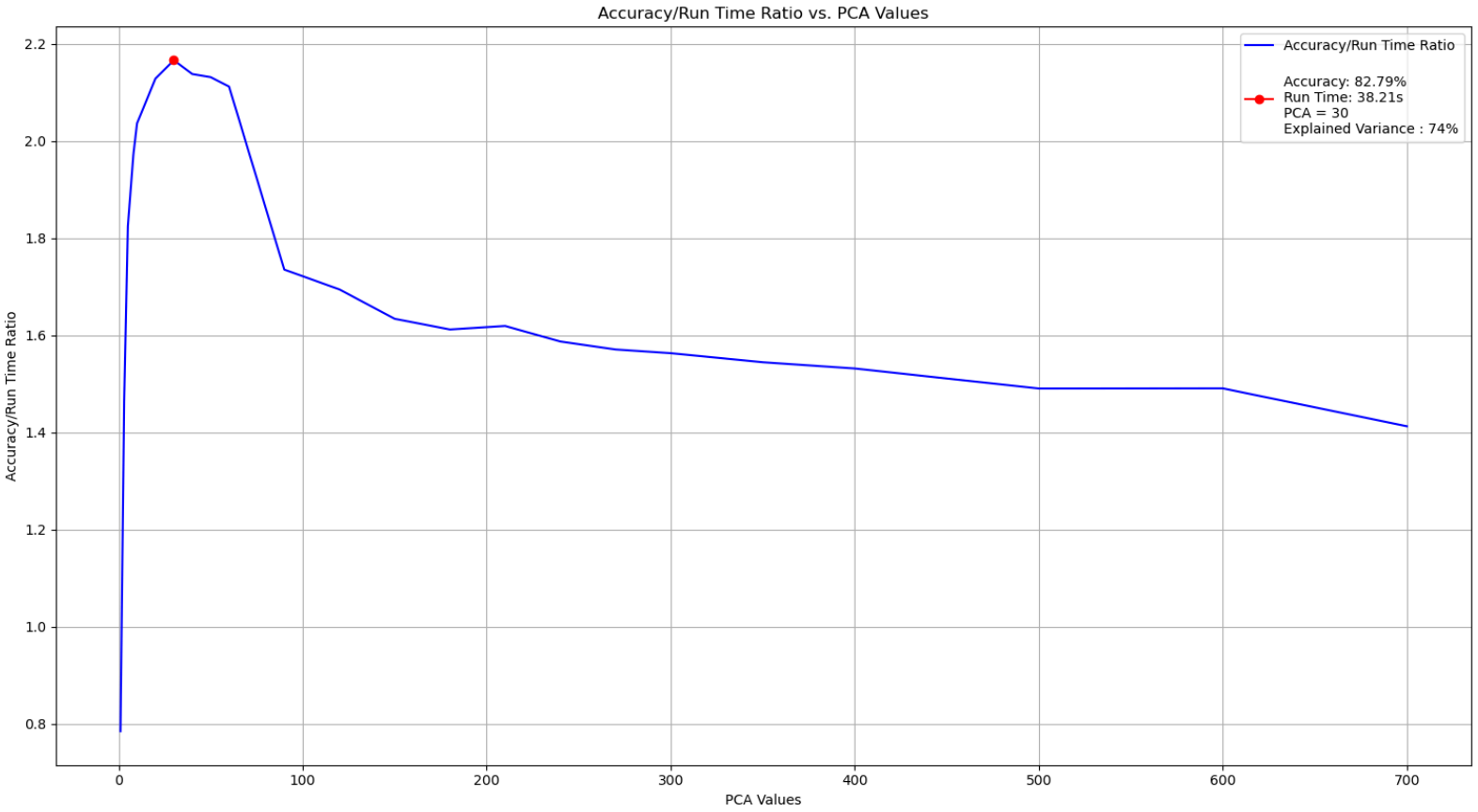
2.Best accuracy for a given learning rate at a fixed max_iter



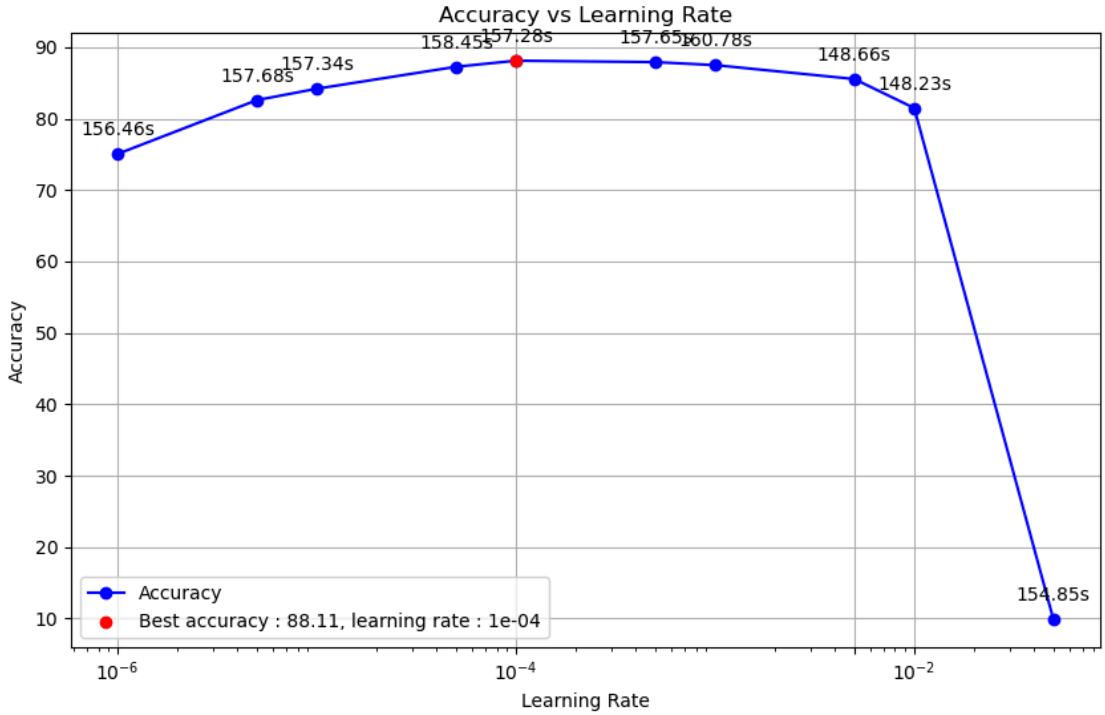
3. PCA values regarding run time and accuracy with most efficient learning rate and max_iters



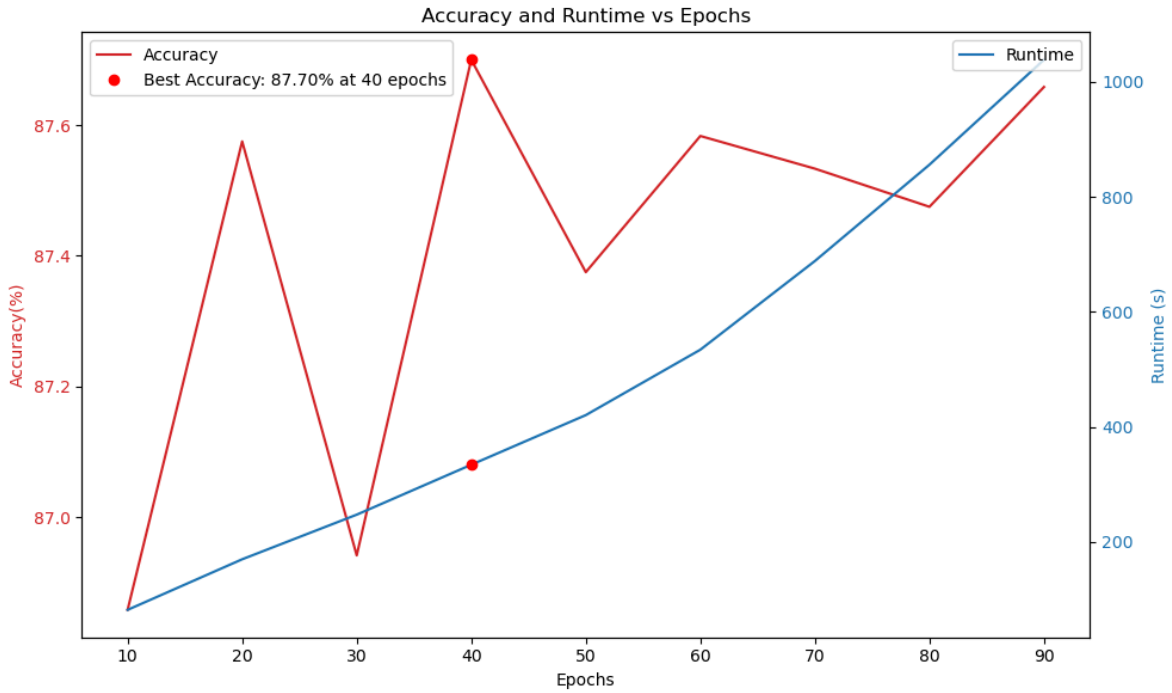
4. PCA values which give trade-off between accuracy and run time



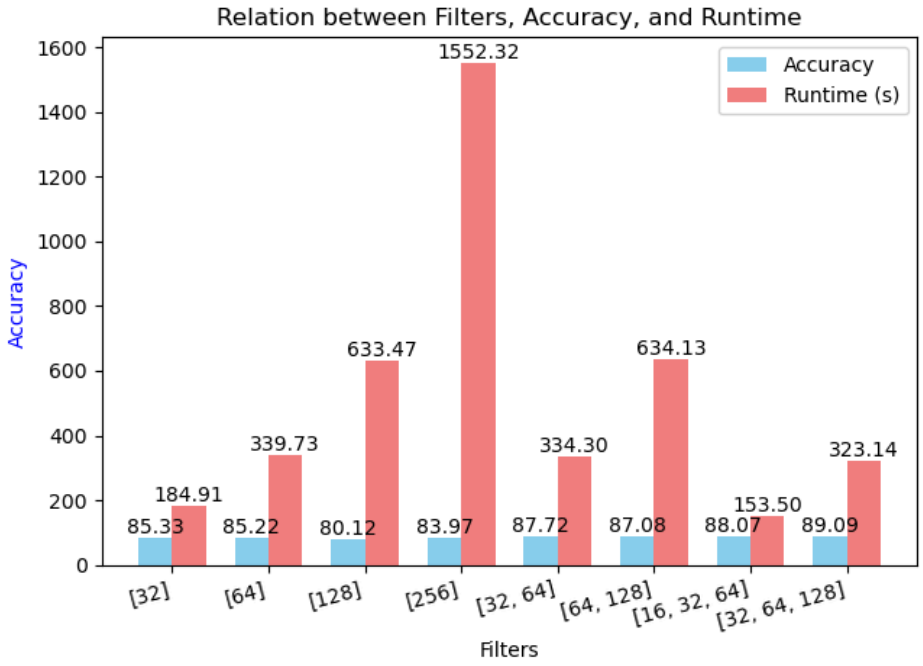
5. Best accuracy for a given learning rate at a fixed max_iters (20) for CNN



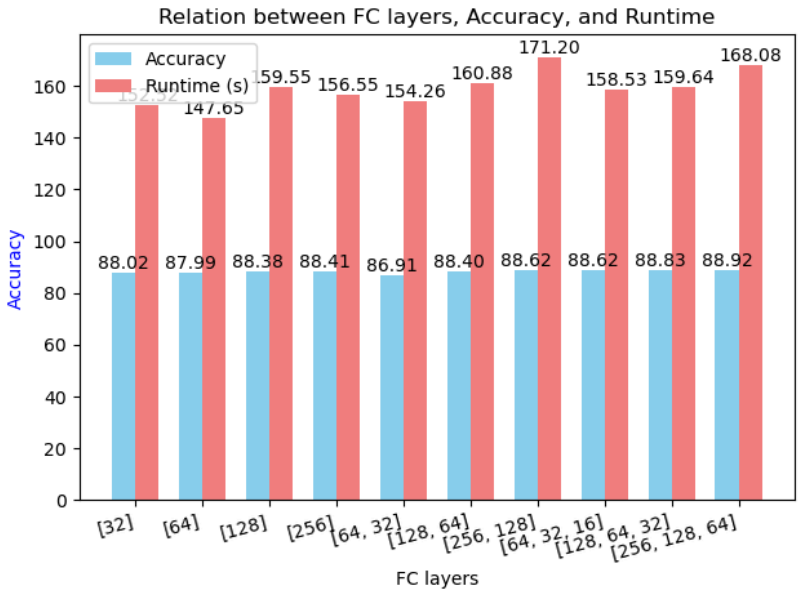
6. Best accuracy for different epochs and their run times (lr = 1e-4)



7. Best accuracy for different sets of fully connected layers and their run times



8. Best accuracy for different sets of fully connected layers and their run times



9. Process for Transformer parameters fine tuning

Test Number	Epochs	learning rate	n_patches	n_bocks	hidden_d	n_heads	nn_batch_siz	accuracy TS	accuracy VS	F1 score TS	F1 score VS	time approx (m
	since it takes a very long time to train we first try with one epoch to eliminate parameters that are completely unappropriate											
1	1	1.00E-05	7	2	8	2	64	0.1003	0.1007	0.0182	0.0183	3
2	1	1.00E-05	7	4	8	2	64	0.1008	0.1012	0.0183	0.0184	5
3	1	1.00E-05	7	2	8	4	64	0.1025	0.096	0.0195	0.0182	3
4	1	1.00E-05	7	2	32	2	64	0.1882	0.1882	0.0656	0.0655	5
5	1	1.00E-05	7	4	32	4	64	0.2551	0.2567	0.1297	0.1314	10
6	1	1.00E-03	7	4	32	4	64	0.5892	0.5921	0.5281	0.5282	10
7	1	1.00E-03	7	2	16	2	64	0.5767	0.5667	0.5258	0.516	3
8	1	1.00E-04	7	2	16	2	64	0.349	0.3483	0.2391	0.2393	3
9	1	1.00E-02	7	2	16	2	64	0.3829	0.3798	0.3054	0.3017	3
10	1	1.00E-03	7	2	8	2	64	0.5784	0.5722	0.5339	0.5308	2
11	1	1.00E-03	14	2	8	2	64	0.4872	0.4862	0.4047	0.4045	3
12	1	1.00E-03	7	2	8	2	16	0.5637	0.5628	0.4965	0.4968	3
13	1	1.00E-03	7	2	8	2	256	0.4016	0.406	0.3095	0.3112	3
14	1	1.00E-03	14	4	128	4	64	0.6019	0.5989	0.5748	0.5735	30
15	1	1.00E-03	7	4	128	4	64	0.6385	0.6327	0.6299	0.6231	20
16	1	1.00E-03	7	4	128	4	64	0.6637	0.6618	0.6363	0.6356	20
17	1	1.00E-03	7	4	128	4	64	0.5045	0.5046	0.4157	0.413	20
	realising it yields quite different results when we train with only one epoch that parameters are now decent we try more epochs despite the time it takes											
18	5	1.00E-03	14	3	64	8	64	0.7455	0.7401	0.7345	0.728	80
19	5	1.00E-03	14	4	64	4	64	0.7588	0.7542	0.7491	0.7434	60
20	5	1.00E-03	7	4	64	4	64	0.7177	0.7162	0.7118	0.7091	40
21	5	1.00E-03	14	4	64	8	64	0.7218	0.7248	0.7221	0.7254	100
22	10	1.00E-03	14	4	64	4	64	0.7805	0.7768	0.7735	0.7696	150
23	20	1.00E-03	7	2	8	2	64	0.7014	0.6992	0.6691	0.6684	30
24	15	1.00E-03	7	4	32	4	32	0.7616	0.7583	0.756	0.7531	70
25	10	1.00E-03	4	6	32	4	64	0.7678	0.7528	0.7605	0.7445	40
26	20	1.00E-03	7	4	64	4	64	0.7612	0.7575	0.7589	0.7567	90