Deep Learning

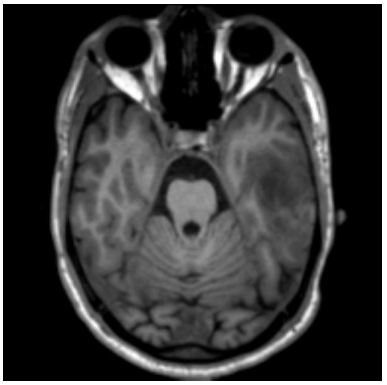
Brain Tumor MRI Classification

Alex-Mihai Serafim

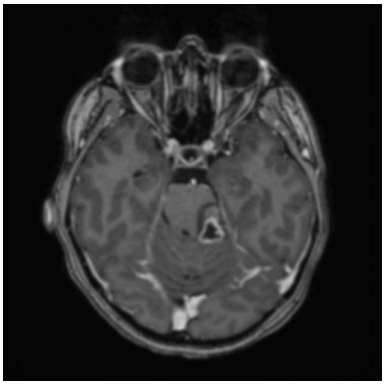
I. Task and dataset description

For this project I chose the Brain Tumor MRI Images, a dataset which contains 4479 total images for 15 different brain tumors. The task is to classify images to the correct brain tumor.

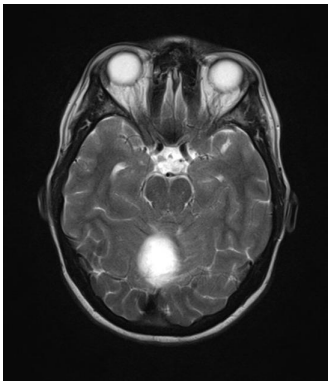
The dataset contains three different types of MRI images: T1, T1C+ (contrast enhanced) and T2. Most types of tumors contain images for all three types of images, so in total we have 44 classes. Each class contained between 75 and 179 images, of size 630 by 630 pixels.



Astrocitoma T1



Astrocitoma T1C+



Astrocitoma T2

A. Image preprocessing

Each image is first resized to 224 by 224 pixels; then only one channel is kept since they are already in greyscale. Next I implemented an image augmentation pipeline: in order to ensure each class has the same number of samples, I set a threshold number of images for each class. Images are randomly selected, random augmentations are applied and the augmented image is added to the dataset. Process repeats until the class hits the threshold of images.

Augmentations are chosen from this list:

- horizontal/vertical flip with a chance of 0.5
- rotation with a max angle of 30°, with a chance of 0.5
- brightness change, with a chance of 0.7
- gaussian noise, with a chance if 0.7

After augmentation, images are normalized to [0, 1].

II. Models

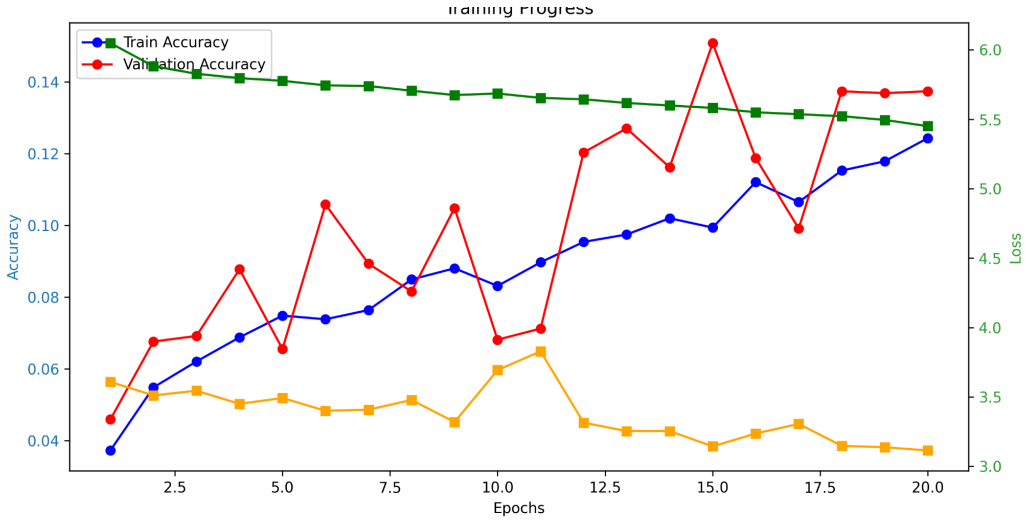
A. Autoencoder

First model tested was an autoencoder. An autoencoder contains two sub modules: and encoder, which takes the input and projects it into a latent subspace. This intermediate projection is then fed into the decoder, whose purpose is to reconstruct the original image. More specifically, I implemented a convolutional autoencoder: I opted for convolutional layers, instead of linear layers. Since the model's aim is to reconstruct the original image, MSELoss is used; the latent space can be considered a form of image embeddings.

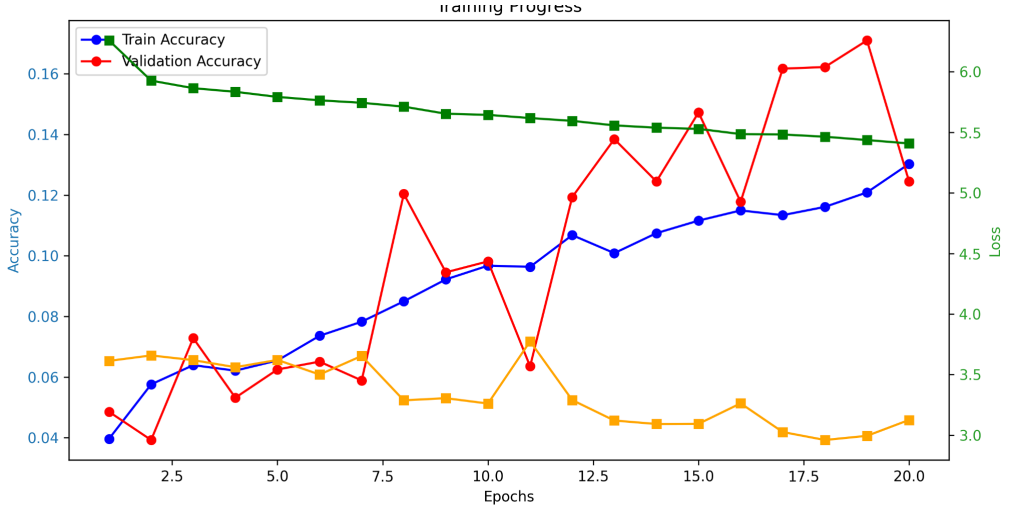
The encoder and decoder each have 4 convolutional layers, followed by a BatchNorm layer and a ReLU activation. A final classifier is used with the image embeddings; the classifier has two fully connected layers. This classifier is trained with CrossEntropyLoss. Both the autoencoder and classifier are trained at the same time, to speed up training time.

To fine-tune the hyperparameters of the model, I trained multiple models on 20 epochs (due to computational constraints) and set the threshold for each class to 220 images. Two hyperparameters were tested: batch size and layer sizes. Below are validation results.

Layer size / Batch size	32	64
[128, 64, 32, 16]	14.10%	13.64%
[256, 128, 64, 32]	17.10%	19.06%



256, 64



128,32

Training loss (green) is higher than validation loss (yellow) because training loss contains reconstruction loss. Validation loss and accuracy is jittery, while the training values are stable. The best configuration is [256, 128, 64, 32] as layer sizes, with a batch size of 64.

B. Vision Transformer

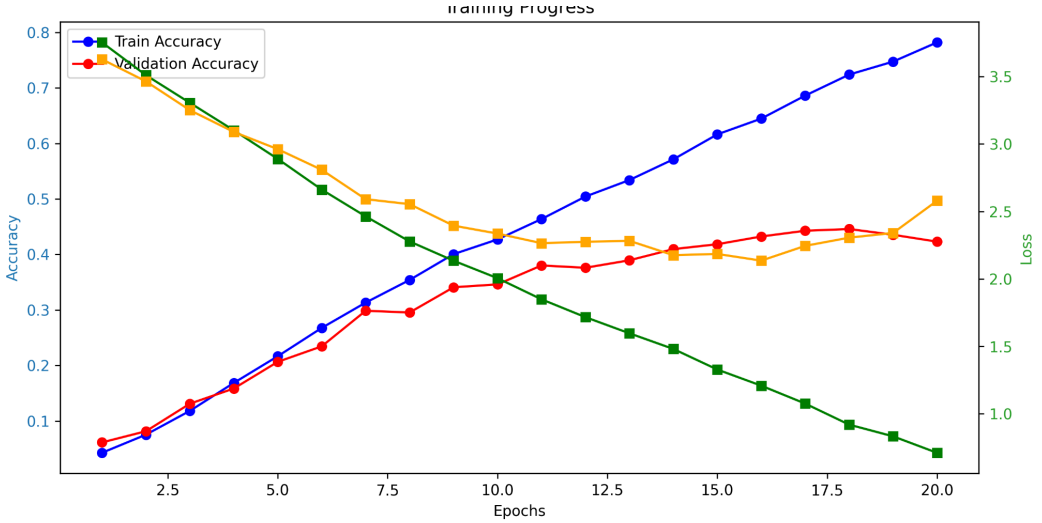
The second model was a ViT, a Transformer model adapted for vision applications. Since Transformers use sequence input, the first step in the ViT is the Patching. Each image is passed through Patching which is just a convolutional layer. After this convolutional layer, we flatten the image to create a sequence.

After Patching, the input goes through Positional Embedding. PosEmb first adds a learnable class token to be used in classification. Then, since Transformers take inputs in parallel, we need a way to encode the position of a certain image patch. The original paper of the Transformer model proposes the sin/cos trick to encode position.

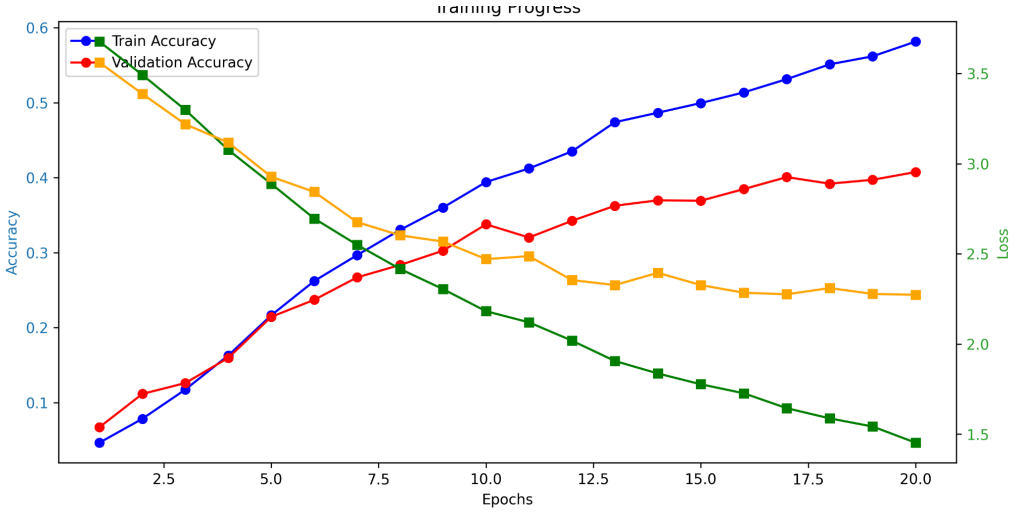
Next, we define the attention head. An attention head is composed of three layers: key, value and query. Query is first multiplied with key to weigh inputs based on their relevance to the current output and the result is multiplied with value to get the final attention. Multiple attention heads are used, resulting in multi-headed attention. The ViT class contains the MHA, two LayerNorms and a final MLP.

Hyperparameters for the ViT were again fine-tuned using 20 epoch-runs. Parameters tuned were the hidden size and batch size.

Hidden dim / Batch size	64	128
128	40.75%	38.69%
256	42.72%	42.30%



256 hidden dim, 128 batch size

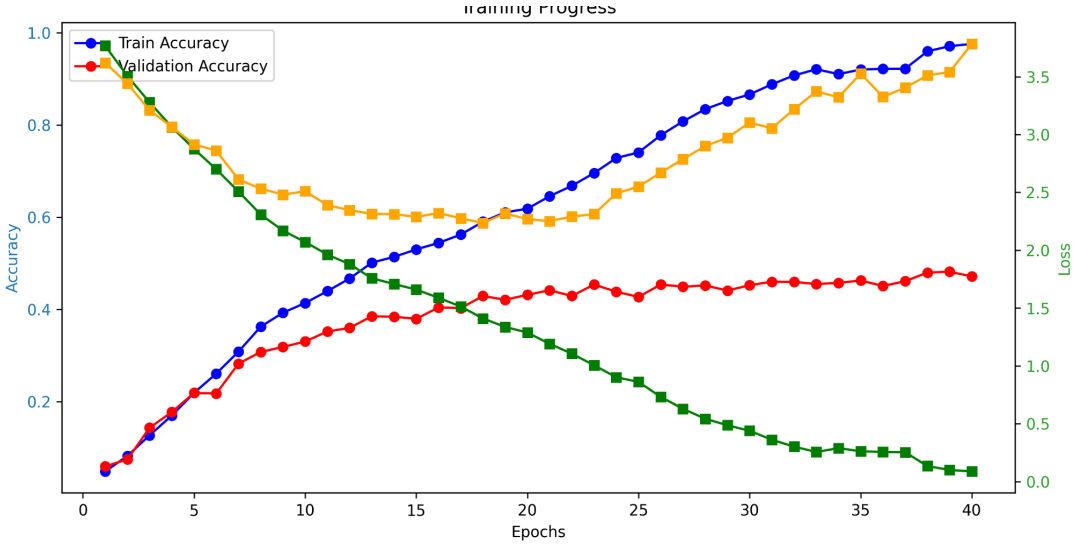


128 hidden dim, 64 batch size

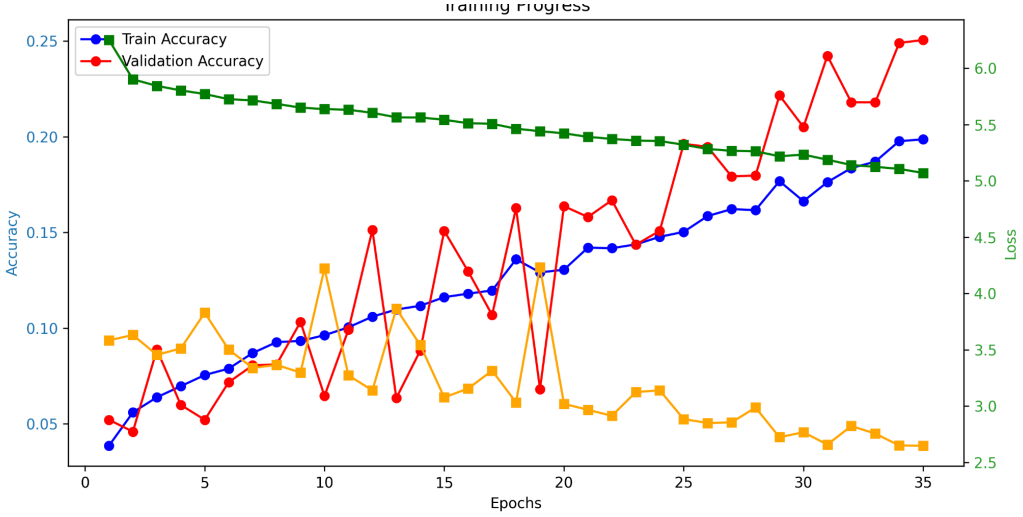
The best configuration is 256 hidden dim with 64 batch size. Here, validation scores are much more stable, however they diverge from the training values after epochs 7-10.

III. Final experiments

In order to test the autoencoder and the ViT, I took the best configurations from the hyperparameter tuning and trained each model for 40 epochs.



ViT



Autoencoder

As we can see, the ViT (48.19%) is significantly more performant than the Autoencoder (25.05%); it is also more stable during training. The Autoencoder would have benefited from more training time.

IV. Conclusion and future work

Both models proved fairly efficient at the given task and they could have been improved mainly by more hyperparameter tuning and more training. One more

interesting thing to note is the data. Since we have three types of medical images for the same diseases, we get triple the number of classes and are adding unnecessary complexity to the system.

The dataset does not provide any sort of id of the images so we cannot correlate images across the different types of images, even though they represent the same disease. If images had patient id labels, then we could group images belonging to the same person and cut the number of classes to 15.

For example, input could be composed of three channels: T1, T1C+ and T2 images. This would allow us to train a more reliable model which could use information from all three types of images at the same time.