Hyperparameter Tuning

Some First Tips

- Try to use **best practices** first:
  - Optimizer: **Adam**
  - Activation: **ReLU**
  - Loss: **Cross Entropy**
- These are "standard choices" for a reason (see lecture!) and work well in most cases
- This helps you to narrow down the search space from the beginning and avoid unnecessary experiments

1. Coarse Random Search

First, we want to get a rough understanding about what value ranges work rather well and which don't at all.

```
best_model, results = random_search(
            dataloaders['train'], dataloaders['val'],
            random_search_spaces = {
                "learning_rate": ([1e-1,1e-7], 'log'),
                "lr_decay": ([0.66, 1], 'float'),
                "reg": ([1e-2, 1e-8], "log"),
                "std": ([1, 1e-7], "log"),
                "hidden_size": ([100,1500], "int"),
                "num_layer": ([2, 6], "int"),
                "activation": ([Relu()], "item"),
                "optimizer": ([Adam], "item"),
                "loss_func": ([CrossEntropyFromLogits()], "item")
             },
             num_search = 500,
             epochs=7, patience=2,
             model_class=MyOwnNetwork)
```

We ran this random search over night and evaluated 500 configurations.

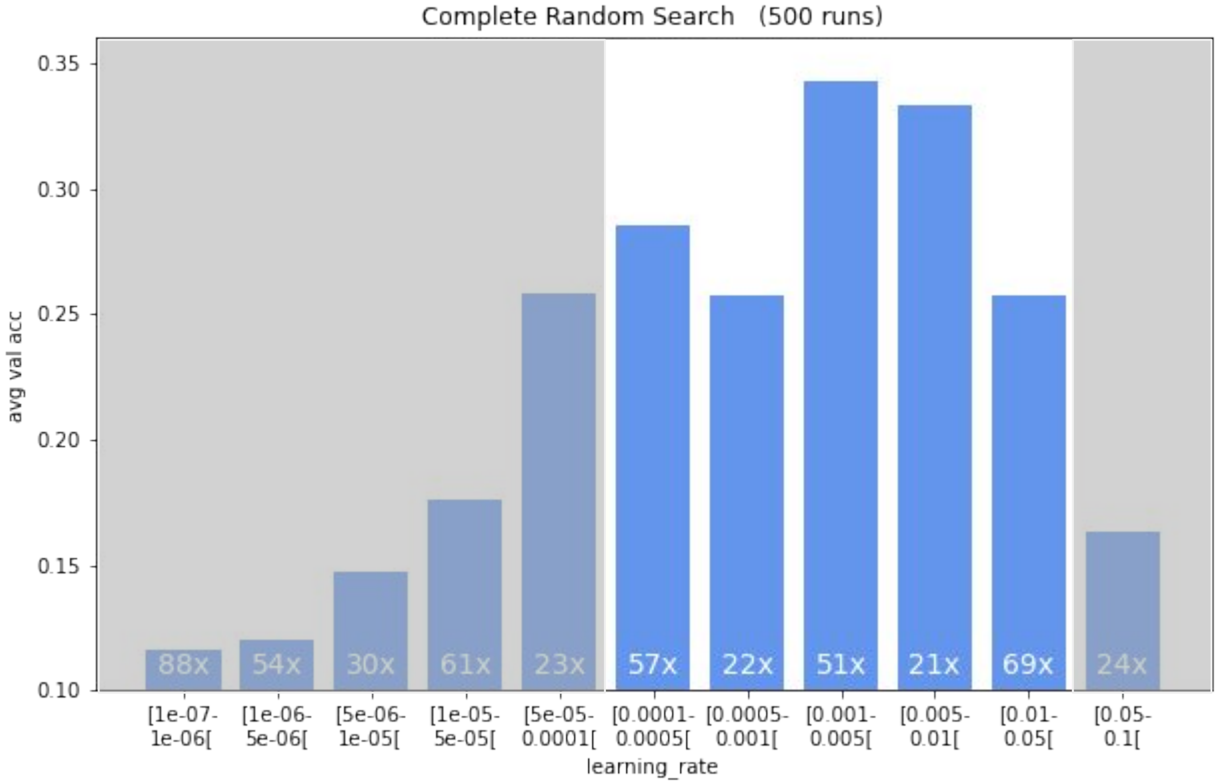
1. Coarse Random Search - Tips

- Search over a broad range of values
- Only train for a small number of epochs
  - Have a look at your training curves. You'll notice that after a certain number of epochs, the loss hardly decreases anymore. Set this as the number of epochs -> saves a lot of time!
- You can train on a subset of your data to save time
  - Make sure it's not too small to get a reasonable approximation of the real performance of your model
- Let the search run for a few hours
  - Still, Hyperparameter Tuning takes a lot of time. And in real projects actually *much* longer than in this exercise, so better get used to it now ;)

Analyze the Results

- Analyze your search results to get a rough understanding of the hyperparameter landscape
  - E.g. Plot Hyperparameter value vs. Accuracy, etc.
  - Compute mean and variance over certain hyperparameter intervals
  - Analyze the models that performed best
  - You can even apply ML techniques, clustering, dimensionality reduction, etc.
- Be careful though
  - Some hyperparameters (like learning rate decay) need more training epochs to have a significant effect on the training/model performance
  - Keep in mind that hyperparameters are highly coupled. E.g. value A might only work well for Hyperparameter 1, if Hyperparameter 2 has value B
  - If certain value-range shows good (or bad) results for *all* the runs, it's a sign that you can select it for your final model (or remove it from the search space)

Learning Rate

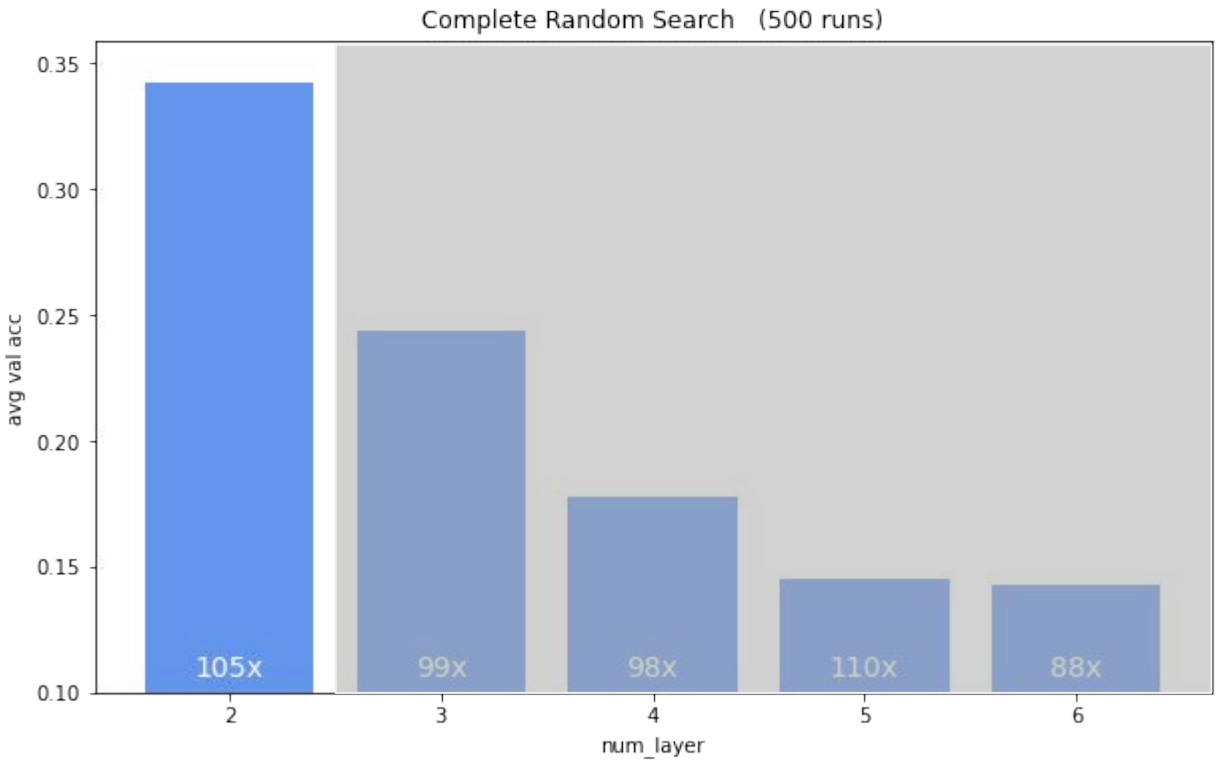


Our results clearly show that very small or very large LRs don't allow any learning at all. We can safely filter out these values to further reduce our search space.

New range:

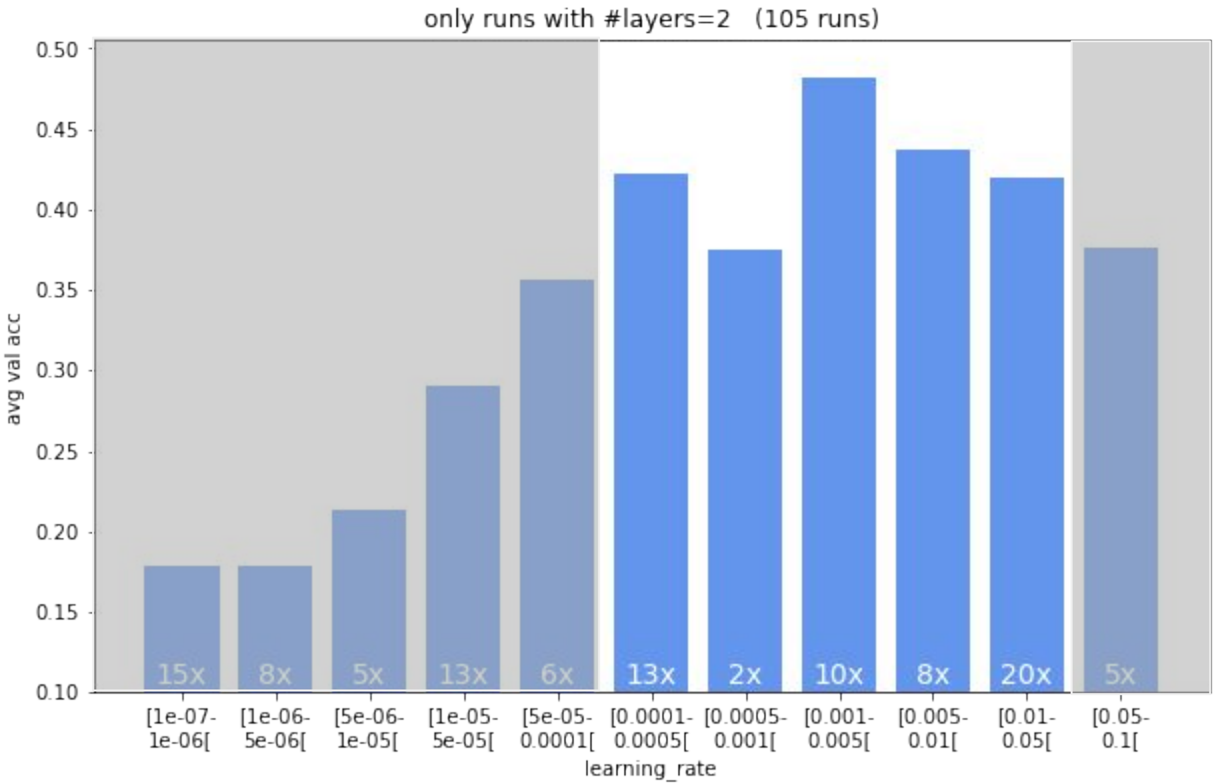
LR = [1e-4, 5e-2]

Number of Layers



It turns out that multiple layers are not really useful here (at least for our simple architecture). Therefore, let's fix the number of **layers to 2**, this also speeds up the further search a lot!

Learning Rate (num_layers=2)



As hyperparameters highly depend on each other, it makes sense to only consider runs with *num_layers==2* for our further evaluations.

Note that since we have much less runs for this case, these results are much less accurate.
Here, we can confirm that pruning low/high values helps.

LR Decay, Reg, Initialization, n_hidden

- As mentioned before, for hyperparameters like **learning rate decay** and **regularization strength** only 7 epochs don't give us a good approximation on their impact on the model, so we need to tune these by searching with more training epochs
- For the **initialization std**, the results don't show any clear patterns, so we keep the whole search space open
  - *Spoiler Alert: In practice you don't need to tune the initialization, as there are methods like He-Initialization that lead to provable better results. You'll learn more about it in the next lecture!*
- The same applies to **N_hidden**. In fact, this hyperparameter depends a lot on the depth of the network, which we now fix to 2, so it makes sense to not make any further decision here.

Best Results

Using only this first coarse random search we already found several models that easily pass the exercise threshold (best val acc: **52.6%**) .

You can gain further insights by analyzing the best performing models. As you can see on the next slides, we can confirm that most of the best models have a low number of layers (blue) and a rather high learning rate (yellow).

Analyzing the best runs

	A	B	C	D	E	F	G	H	I	J	K
1	run	val-acc	learning_rat	log(lr)	lr_decay	reg	log(reg)	std	log(std)	hidden_size	num_layer
2	2020-12-17-01:47:09	0,52604167	0,00207216	-2,6835762	0,69001942	6,28E-06	-5,2023365	9,98E-07	-6,0007428	1364	2
3	2020-12-16-13:23:45	0,51642628	0,00202942	-2,692629	0,75824335	4,98E-07	-6,3024598	0,00020451	-3,6892883	473	2
4	2020-12-17-00:03:12	0,51482372	0,01161251	-1,9350739	0,6607703	5,64E-05	-4,2490281	0,04361678	-1,3603464	983	2
5	2020-12-16-20:13:25	0,50891426	0,00045071	-3,3460981	0,80779747	1,74E-06	-5,758609	0,00935924	-2,0287594	932	2
6	2020-12-16-16:45:45	0,49889824	0,00024626	-3,6086106	0,90894699	0,00013087	-3,8831508	0,00101984	-2,9914669	1424	2
7	2020-12-17-12:44:03	0,49679487	0,00037502	-3,4259477	0,95293837	3,12E-07	-6,5056069	0,03101998	-1,5083585	1497	2
8	2020-12-16-16:29:18	0,49439103	0,0222987	-1,6517205	0,67256411	9,70E-05	-4,0134369	7,53E-07	-6,1230921	807	2
9	2020-12-16-22:09:00	0,49429087	0,00030072	-3,5218394	0,84110456	5,31E-07	-6,2748049	8,95E-05	-4,0480519	1355	2
10	2020-12-16-23:40:58	0,49419071	0,00103719	-2,9841419	0,94883268	2,01E-09	-8,6960177	0,01262235	-1,8988597	240	3
11	2020-12-16-20:31:44	0,4926883	0,00175088	-2,7567446	0,96386164	1,08E-06	-5,9662878	0,00102239	-2,9903841	463	2
12	2020-12-16-13:48:20	0,4916867	0,00537997	-2,2692199	0,89261903	2,61E-05	-4,5834461	0,00132461	-2,8779125	285	2
13	2020-12-17-10:29:04	0,49128606	0,01647803	-1,7830947	0,74853715	2,03E-08	-7,6932493	0,01077647	-1,9675234	1134	2
14	2020-12-17-12:02:33	0,49128606	0,00305564	-2,5148977	0,89742436	0,00013121	-3,882024	2,10E-07	-6,6768114	234	2
15	2020-12-17-09:51:58	0,48988381	0,00116807	-2,9325306	0,82800218	4,70E-06	-5,3282827	0,00058372	-3,2337965	874	3
16	2020-12-17-11:13:13	0,48968349	0,00021446	-3,6686493	0,89390911	6,34E-08	-7,197886	5,71E-05	-4,24355	1312	2
17	2020-12-16-22:25:52	0,48858173	0,02238176	-1,6501058	0,72917956	2,02E-07	-6,6941047	0,00160335	-2,7949713	971	2
18	2020-12-17-06:03:21	0,48577724	0,00045201	-3,344856	0,77016255	2,27E-08	-7,6447285	6,55E-05	-4,183871	629	2
19	2020-12-16-16:25:30	0,48557692	0,00300663	-2,5219203	0,87221038	0,00058888	-3,2299707	0,00101997	-2,9914144	702	2
20	2020-12-16-20:23:59	0,48277244	0,02770224	-1,557485	0,69865014	4,76E-06	-5,3225584	0,00010675	-3,9716429	380	2
21	2020-12-17-09:55:31	0,48217147	0,00183616	-2,7360902	0,99321012	3,74E-08	-7,426786	0,00327585	-2,484676	870	2
22	2020-12-17-13:30:09	0,47996795	0,00110646	-2,9560653	0,69865574	0,00367965	-2,4341938	5,67E-06	-5,246554	165	2
23	2020-12-16-22:46:32	0,47946715	0,00595508	-2,2251127	0,79772423	0,00126358	-2,898397	0,02001093	-1,6987327	666	2
24	2020-12-16-17:40:51	0,4770633	0,00322104	-2,4920041	0,97874093	2,65E-09	-8,5763162	7,78E-07	-6,1087813	115	2
25	2020-12-17-09:45:01	0,47676282	0,0008435	-3,0739127	0,78147951	1,18E-08	-7,9284791	0,00027408	-3,5621267	428	3
26	2020-12-16-17:20:27	0,47445913	0,00490045	-2,3097638	0,91893944	1,20E-09	-8,9213197	2,44E-05	-4,6119894	881	2
27	2020-12-17-12:13:59	0,47395833	0,0014251	-2,8461549	0,90862507	5,01E-06	-5,300213	0,04958649	-1,3046366	659	3
28	2020-12-16-19:20:01	0,47325721	0,00166617	-2,77828	0,82176662	1,41E-06	-5,8516311	1,56E-07	-6,8055029	1443	3
29	2020-12-17-05:01:04	0,47245593	0,00225076	-2,64767	0,89821977	0,00117435	-2,9302033	0,00037223	-3,429194	1478	3
30	2020-12-16-12:20:28	0,4672476	8,71E-05	-4,0597801	0,96969425	9,39E-07	-6,0275084	0,00071438	-3,1460702	1187	2

Analyzing the worst runs

	A	B	C	D	E	F	G	H	I	J	K
470	2020-12-16-23:32:58	0,09975962	2,29E-07	-6,6396095	0,93661008	3,78E-06	-5,4222594	0,6061806	-0,217398	188	4
471	2020-12-17-00:31:57	0,09955929	9,49E-07	-6,022735	0,91602552	0,00022882	-3,6405015	0,47763719	-0,3209019	248	3
472	2020-12-17-15:06:48	0,09945913	1,99E-07	-6,7006671	0,82740503	1,57E-08	-7,8030634	5,87369677	0,76891152	860	5
473	2020-12-16-23:19:10	0,09885817	0,0200412	-1,6980762	0,90602403	1,24E-07	-6,9071079	0,11950815	-0,9226025	769	4
474	2020-12-17-07:07:25	0,09875801	2,08E-07	-6,6809304	0,98141971	0,00947225	-2,0235467	0,00014246	-3,8462994	1469	3
475	2020-12-17-14:40:26	0,09865785	1,06E-06	-5,9751233	0,71463285	0,0001575	-3,8027211	0,67738142	-0,1691667	827	5
476	2020-12-16-23:55:50	0,09845753	1,02E-05	-4,9900024	0,82846439	2,60E-05	-4,5857276	0,73173649	-0,1356453	1363	4
477	2020-12-16-13:50:02	0,09835737	1,13E-05	-4,9457326	0,78875986	1,24E-06	-5,9055152	0,00015106	-3,8208591	118	3
478	2020-12-16-22:00:22	0,09835737	1,05E-07	-6,9769223	0,74792408	1,35E-09	-8,8697952	0,00031121	-3,5069436	1231	6
479	2020-12-17-03:05:51	0,09835737	7,59E-06	-5,1196488	0,71710447	3,62E-08	-7,4407465	7,03E-05	-4,1528347	258	6
480	2020-12-17-03:22:16	0,09835737	0,00013996	-3,8540091	0,72360442	0,00012069	-3,9183159	1,22E-05	-4,9141106	1209	6
481	2020-12-17-10:58:58	0,09835737	0,00051248	-3,2903259	0,83163021	2,08E-08	-7,6813577	7,98E-07	-6,0980523	774	4
482	2020-12-16-14:02:03	0,09825721	5,34E-05	-4,2723385	0,79393612	1,69E-06	-5,7728356	1,33E-05	-4,8760074	1440	4
483	2020-12-16-18:36:05	0,09825721	3,89E-05	-4,4101405	0,91173747	3,50E-08	-7,456148	7,18E-06	-5,1436968	150	4
484	2020-12-16-19:45:26	0,09825721	4,81E-07	-6,3178305	0,67420016	5,19E-06	-5,2849814	5,59E-07	-6,2529327	834	4
485	2020-12-17-01:29:08	0,09825721	4,66E-06	-5,3317615	0,68998048	1,40E-08	-7,8528314	8,36E-05	-4,0779499	589	3
486	2020-12-17-12:06:43	0,09825721	2,89E-06	-5,5389365	0,77943228	4,44E-05	-4,3524699	0,00034801	-3,4584048	313	4
487	2020-12-16-19:47:45	0,09815705	0,00028881	-3,5393816	0,95221812	2,27E-06	-5,6448587	1,37E-07	-6,8641076	1047	5
488	2020-12-17-04:07:39	0,09815705	2,80E-05	-4,5534466	0,94761764	3,69E-08	-7,4327394	2,26E-05	-4,6468406	673	5
489	2020-12-17-03:00:00	0,09755609	1,80E-06	-5,7447874	0,85237893	0,00017419	-3,758979	0,08553933	-1,0678341	264	4
490	2020-12-17-09:04:57	0,09605369	3,75E-06	-5,4258861	0,72874852	5,84E-07	-6,2333692	0,18218168	-0,7394953	424	6
491	2020-12-16-23:22:31	0,09595353	9,50E-07	-6,0224496	0,79746229	1,04E-05	-4,9830062	0,18327136	-0,7369054	569	6
492	2020-12-17-11:36:51	0,09415064	1,78E-07	-6,7485874	0,72246593	8,19E-05	-4,0867671	0,00581711	-2,2352931	482	5
493	2020-12-17-05:48:03	0,0922476	4,52E-07	-6,3444047	0,99045627	0,00025394	-3,5952712	4,14191731	0,61720142	1378	5
494	2020-12-16-23:28:43	0,09094551	2,44E-07	-6,6118364	0,80286761	0,0001526	-3,8164576	0,33805211	-0,4710164	1371	5
495	2020-12-17-10:24:07	0,09034455	2,25E-07	-6,6484686	0,67667693	0,00192537	-2,7154862	3,54182892	0,54922758	397	5
496	2020-12-16-22:37:06	0,08894231	4,04E-07	-6,3931858	0,67247489	7,09E-05	-4,1495649	0,58548474	-0,2324844	238	4
497	2020-12-17-03:32:46	0,08854167	2,11E-07	-6,675568	0,94369672	2,87E-06	-5,5427309	0,13619178	-0,8658491	1018	5
498	2020-12-17-15:14:41	0,08834135	2,29E-06	-5,6393292	0,91753655	6,24E-06	-5,2051032	0,25005902	-0,6019575	729	5
499	2020-12-17-04:27:23	0,08673878	2,96E-06	-5,5287354	0,90333627	1,08E-05	-4,9684495	1,7237435	0,23647264	1444	6
500	2020-12-17-06:49:07	0,0766226	1,26E-07	-6,8994997	0,76328113	3,11E-07	-6,5077486	0,0090492	-2,0433899	1403	5
501	2020-12-17-10:09:35	0,07161458	5,88E-07	-6,2307619	0,95369212	1,48E-07	-6,828516	7,29308366	0,8629112	271	2

2. Run a more narrowed down search

- Now that we added some constraints to our search space, run another search with a **larger number of epochs** to refine the results
- If your search space is now small enough, a grid search can be a good choice to systematically explore the remaining search space.
- Otherwise, simply run another random search
- Note that parameter tuning is an iterative process and most often you run multiple searches until you find a good set of hyperparameters

2. Results

- Our narrowed down random search gives us a best model that already achieves **55.3%** validation accuracy.
- It now also comes clear that the best models have a rather high number of units. This makes sense due to the low number of layers.
- If you want, you can further adjust the search space and run additional searches.

2nd Search - Best Models

	A	B	C	D	E	F	G	H	I	J	K
1	run	val-acc	learning_rat	log(lr)	lr_decay	reg	log(reg)	std	log(std)	hidden_size	num_layer
2	2020-12-18-06:43:15	0,55308494	0,02789742	-1,5544359	0,72614584	0,0003007	-3,5218686	0,03743166	-1,4267609	1094	2
3	2020-12-18-12:26:27	0,54156651	0,00413545	-2,3834774	0,74789975	9,76E-05	-4,0106552	1,89E-06	-5,7230802	1079	2
4	2020-12-18-07:28:30	0,53926282	0,00437007	-2,3595116	0,73748696	1,11E-05	-4,9556309	1,14E-05	-4,9436193	740	2
5	2020-12-18-08:07:37	0,53886218	0,00255097	-2,5932942	0,81147093	2,38E-05	-4,6236853	1,32E-05	-4,8797183	1034	2
6	2020-12-18-09:04:20	0,53816106	0,00611225	-2,2137991	0,72771553	1,18E-08	-7,9292987	3,31E-07	-6,4805505	754	2
7	2020-12-18-08:28:47	0,53735978	0,01377831	-1,8608041	0,64992265	2,82E-07	-6,5492904	0,02810201	-1,5512626	1191	2
8	2020-12-18-03:35:17	0,53655849	0,00125426	-2,9016127	0,75746033	0,0001016	-3,9931165	0,00087992	-3,0555545	1298	2
9	2020-12-18-00:36:20	0,53635817	0,00141643	-2,8488061	0,79983747	1,05E-06	-5,9778659	1,06E-07	-6,9753478	1249	2
10	2020-12-18-12:43:57	0,53635817	0,00705319	-2,1516144	0,65048257	1,84E-07	-6,7353499	2,11E-07	-6,6765297	793	2
11	2020-12-18-03:44:53	0,53625801	0,00682602	-2,1658323	0,58969416	2,12E-05	-4,6730267	0,00040044	-3,3974576	1189	2
12	2020-12-17-17:54:05	0,53605769	0,00614675	-2,2113541	0,77532188	5,42E-07	-6,2659591	5,65E-06	-5,2477959	1270	2
13	2020-12-17-23:05:34	0,53605769	0,00153203	-2,8147331	0,83296462	3,60E-05	-4,4439556	2,30E-05	-4,637994	1069	2
14	2020-12-18-05:14:23	0,53605769	0,01337081	-1,8738424	0,77450042	2,71E-06	-5,5667746	0,00016135	-3,7922238	1225	2
15	2020-12-18-02:23:33	0,53545673	0,0015771	-2,802141	0,71887584	1,45E-06	-5,8395103	0,00037122	-3,4303738	1261	2
16	2020-12-18-02:33:30	0,53545673	0,00241609	-2,6168875	0,71825898	1,90E-06	-5,721343	1,59E-05	-4,7994565	689	2
17	2020-12-18-04:18:44	0,53545673	0,00204384	-2,6895533	0,75851666	0,00062334	-3,205275	0,01961616	-1,707386	567	2
18	2020-12-18-05:25:39	0,53525641	0,01483656	-1,8286668	0,67022193	1,70E-07	-6,7707082	0,001919	-2,7169258	1068	2
19	2020-12-18-14:10:07	0,53515625	0,01581453	-1,8009438	0,63817815	1,94E-05	-4,7128368	0,00067363	-3,1715779	926	2
20	2020-12-18-11:18:28	0,53485577	0,00445611	-2,3510443	0,62682384	6,43E-05	-4,1914678	0,00233318	-2,6320513	895	2
21	2020-12-18-12:14:20	0,53475561	0,00135514	-2,8680154	0,88105163	0,00013109	-3,8824298	1,76E-07	-6,7547717	1146	2
22	2020-12-18-11:04:57	0,53465545	0,00130494	-2,8844102	0,79332312	3,09E-05	-4,5093828	0,00337411	-2,4718405	837	2
23	2020-12-17-20:27:42	0,53315304	0,00079508	-3,0995914	0,89268099	8,23E-05	-4,0846601	2,85E-06	-5,5448535	971	2
24	2020-12-18-09:51:21	0,53285256	0,00048415	-3,315024	0,88490571	0,00023682	-3,6255807	0,02058569	-1,6864347	1115	2
25	2020-12-17-19:26:53	0,53265224	0,01024545	-1,989469	0,5213646	0,00046439	-3,3331159	0,00018339	-3,7366212	990	2
26	2020-12-17-18:15:10	0,53255208	0,01329896	-1,8761824	0,57712308	3,16E-06	-5,5007734	2,44E-07	-6,6122821	992	2
27	2020-12-18-11:11:37	0,5322516	0,00506766	-2,2951927	0,81027963	0,00051124	-3,2913749	0,0026012	-2,5848262	285	2
28	2020-12-18-09:21:18	0,53215144	0,0437676	-1,3588472	0,54681401	0,00023464	-3,6295903	0,02716591	-1,5659757	779	2
29	2020-12-18-13:35:20	0,53195112	0,00139985	-2,8539198	0,72429729	4,92E-05	-4,3080351	4,06E-07	-6,3916675	1004	2
30	2020-12-18-01:47:26	0,53185096	0,01423128	-1,8467559	0,75991589	4,59E-08	-7,3384852	0,00014816	-3,8292731	1262	2

3. Use Data Augmentation!

An easy way to further improve our performance is to apply data augmentation. Even simply augmentations like mirroring can significantly boost your performance!

By **Flipping** the image our final test accuracy is **56.76%.**

Score	Pass
56.76	✔

Feel free to further push the performance, by adding more augmentations, improving your architecture (e.g. different numbers of units per layer) or searching longer!

Lessons Learned

Hyperparameter Tuning is challenging

- The search space grows exponentially with the number of hyperparameters tuned
- Hyperparameters have highly non-linear behavior and influence each other a lot
- For every configuration, a model needs to be trained, predictions must be computed on the validation data, and the evaluation metric must be evaluated => finding a good models takes a LOT of computational power and time!
- Note: while using a GPU for this simple dataset will speed computations up a lot, in the future your data and models will become much more complex, so searches will even take longer!
- This makes hyperparameter tuning one of the main challenges in DL!

Also, note that there is no "one-fits-all"-solution. However, we hope that this case study gave you some good ideas of how you can approach such a task.

More Tricks

There are further tricks that help you make hyperparameter tuning more efficient:

- **Trial Pruning**
- **Parallelization**
- **Bayesian Optimization**
- **...**

Pruning & Parallelization

Trial Pruning

- Early Stopping is already a good approach to avoid unnecessary epochs
- However, there are more sophisticated approaches to detect and cancel unpromising trials, such as **Median Pruning**: if at any epoch n the current evaluation metric (e.g., validation accuracy) is lower than the **Median** of all previously executed runs: consider the current run as unpromising and cancel it

Parallelization

- You can parallelize your hyperparameter optimization over multiple threads, processes, GPUs, or servers

Bayesian Optimization

Bayesian Optimization

- Grid-search and random search do not take into account the outcome of previous results (so far we did this manually by analyzing the results and constraining the search space)
- *E.g. if a learning rate of 1e-7 is sampled 10 times and always leads to bad results, you wouldn't try it further - but random search doesn't consider this*
- **Bayesian optimisation** on the other hand takes into account past evaluations when choosing the hyperparameter-set to evaluate next -> saves time by filtering out non-promising subsets of the hyperparameter space

Hyperparameter Tuning Frameworks

There are many frameworks that help you automate your hyperparameter optimization and make it more efficient.

One good choice is **Optuna**.

- It's an open-source hyperparameter optimization that provides support for all previously mentioned features.
- We'll also try to provide a jupyter-notebook for you to try it out!

Initialization

- Glorot Initialization: Initialized weights in network by drawing from normal distribution with 0 mean and var:

$$Var(w_i) = \frac{1}{fan\_in}$$

Fan_in: number of incoming neurons / input units for weight tensor

- If using ReLU activation, **He-Initialization** turns out to be the best choice:

$$Var(w_i) = \frac{2}{fan\_in}$$

Numpy:

```
W[l] = np.random.randn(layer_dims[l-1],
                        layer_dims[l])\
        * np.sqrt(2/layer_dims[l-1])
```