# Project Updates

The final project pipeline should look like the following:

1. Fine-tuning

2. Hyperparameter-tuning

3. Testing

1. **NEW:** Fine-tuning of the model with a dataset that is just used for fine-tuning, split into training and validation. During fine-tuning a standard set of hyperparameters is used and all layers except the final layer (classification layer) of the network are freezed so that only the classification layer is trained on the new data.

Fine-tuning is done for each of the five models and the best model/best fine-tuned weights for each model is saved.

Important: seed must be set!

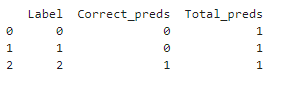
1. **Updated:** Hyperparameter tuning of the best models from step 1 (so the saved model/models must be then used) with another dataset, which is also split into training and validation set.

* *During hyperparameter-tuning several combinations of hyperparameters should be evaluated: batch size, learning rate, epochs, optimizer*. This should be done for all five transformer models. It should automatically save the best model/combination of hyperparameters for each model.

For example: we want to make a run only with model1 and different hyperparameter options. Best combination should be saved (should not overwrite saved model/weights from step 1). Then we would run model2 with different hyperparameter options and that best combination is saved next to the saved model1 and so on. In case we run model1 again, the previously saved model1 should be overwritten.

Important: seed must be set.

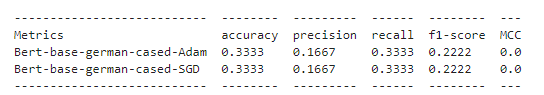
* The overview of correct preds and total preds



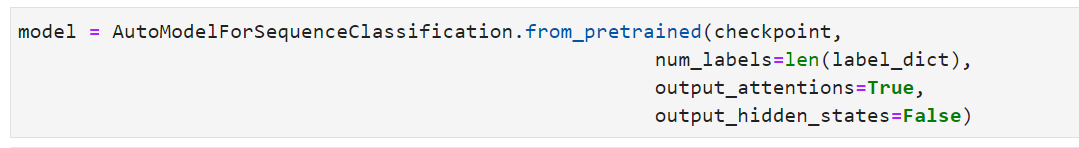
should be *saved in a global variable*, which can be run after the hyperparameter tuning code.

* The overview of the models results should be *expanded so that besides model name and optimizer also the other hyperparameter options* should be included like

Bert-base-german-cased-adam-epoch3-lr5e-5-batchsize16



* Both, the fine-tuning and the hyperparameter tuning pipeline should include two alternative codes for the model setup (that we can choose from), one with a custom model head (which already exists) and one, that automatically instantiates a model head for sequence classification via the huggingface AutoModel Class (does not exist yet):



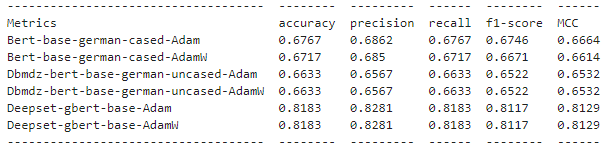
1. **Updated:** Testing the best model/parameter combination (one saved model for each of the 5 models) from step 2 on a testset.

For testing we need the following output information:

* Mapping of the label number to the label names 🡪 **A csv file with the label numbers in one column and the corresponding label text is send**

In the mismatched\_df not only the label numbers should be displayed but also the label names (for the correct label and the predicted label).

* At the moment the mismatch\_df does not contain only the mismatches but also the correct samples.
* Overview of the metrics like it is implemented for training/validation:

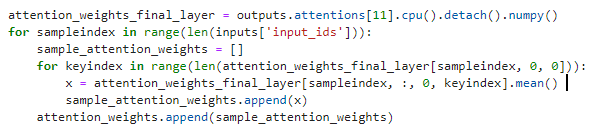


* Attention weights of *the last layer and sum over all heads for each token*🡪 Example implementation found in this source:

[*https://www.linkedin.com/pulse/visualizing-berts-self-attention-layers-help-explain-its-hasan*](https://www.linkedin.com/pulse/visualizing-berts-self-attention-layers-help-explain-its-hasan)

*“The weights being visualized are the weights associated with the [CLS] (the token used to make predictions) at the last self-attention layer which have been normalized across the 12 attention heads (by token) and then across the tokens. Most of the resulting weights are very small (less than 10e-3) so it is hard to perceive the relative importance of each token when they are highlighted based on its weight value. To remedy this, I divided the set of weights for each prediction by its maximum value so that the darkest highlighted word is the most relatively important word and the next darkest is the next relatively important, and so on.“*

Sample code snippet that might help for implementation.



Visualized like in the source above:

