

Model documentation "Genetic Engineering Attribution Challenge" (2nd place, Team: fit_dna)

1) Who are you (mini-bio) and what do you do professionally?

ZFTurbo: My name is Roman Solovyev. I live in Russia, Moscow. I'm Ph.D in the microelectronics field. Currently I'm working in the Institute of Designing Problems in Microelectronics (part of Russian Academy of Sciences) as a Leading Researcher. I often take part in machine learning competitions. I have extensive experience with GBM, Neural Nets and Deep Learning as well as with development of CAD programs in different programming languages (C/C++, Python, TCL/TK, PHP etc).

bragin: My name is Ivan Bragin. I live in Voronezh, Russia. I have a lot of experience in developing high-load services using technologies such as Hadoop, Spark, Cassandra, Kafka. Three years ago, I changed my specialization to machine learning. For two years, I worked as a developer of computer vision algorithms for embedded devices. Currently I'm developing a recommendation system in the Russian social network.

2) What motivated you to compete in this challenge?

ZFTurbo: The problem looked very interesting. I also wanted to play with some Conv1D neural nets.

bragin: Clear input data and simple baseline. When I reviewed it I already had a lot of ideas and could not stop.

3) High level summary of your approach: what did you do and why?

1. Data analysis

The metadata is valuable information but we did not find any insights there. The sequences are even more valuable and after analysis we found some interesting things.

Two DNAs generated in the same lab-of-origin often contain similar subsequences. These matched subsequences are the main features that can be extracted from DNA sequences. We decided to search for important subsequences instead of analyzing a sequence as a sentence. This insight helped us to follow the true way and build efficient architecture.

Some provided sequences are read 5'→3' direction but others are read 3'→5' direction. It was detected based on Blast output. This insight helped us to improve the quality of models.

2. Model development

2.1 Simple model architecture

Our main idea was to find popular subsequences which describe a lab-of-origin. We tested 2 approaches:

1. Use DNN for automatic sequence analysis.

We build a DNN architecture which automatically extracts subsequences and classifies them. The architecture is presented in Figure 1.

2. Based on Blast output.

If Blast finds a similar subsequence of any pair of DNAs we cut it and add to the set of interesting subsequences. Then we build a matrix test+train size to number of sequences and it is about 80000 to 10^7 . This is too big for training. After filtering only popular subsequences and applying Principal Component Analysis (PCA) we extracted 1024 main components and successfully used them to build the lab-of-origin classifier.

There are issues in this approach. Blast doesn't find all matches in a sequence pair, it reports only the longest one. Analysis of 10^7 subsequences requires a lot of hardware resources so we have to reduce it significantly and so lose a lot of information.

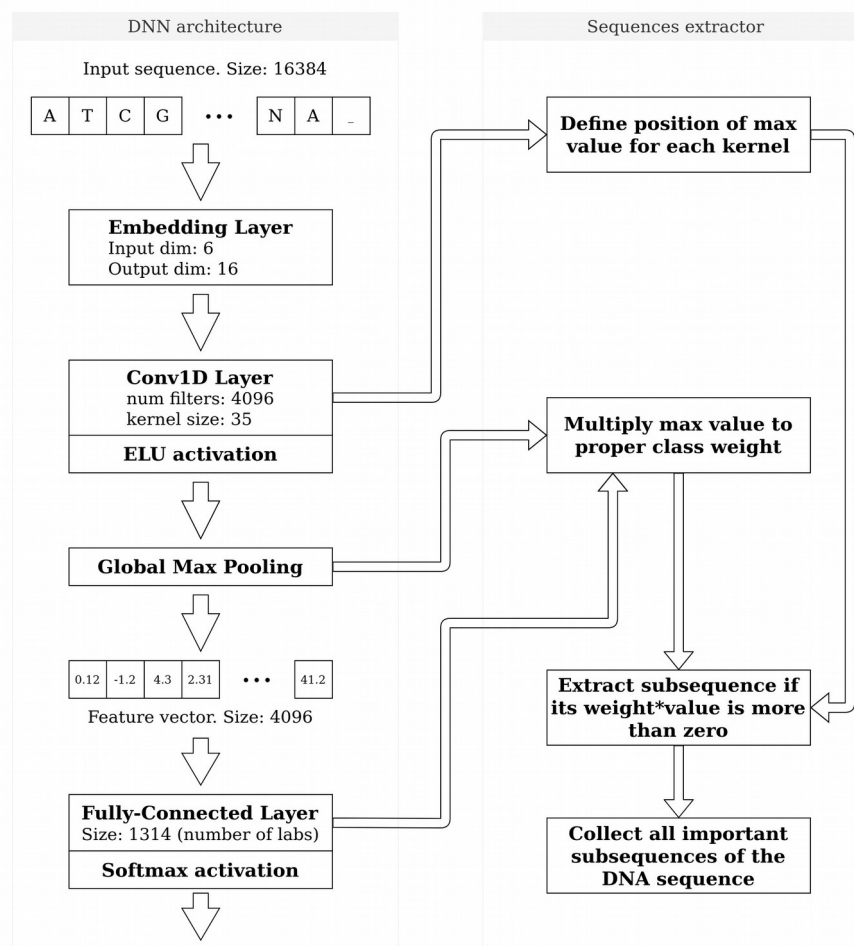


Figure 1. End-to-end DNN model which gets DNA sequence “as is” and returns the vector of lab-of-origin probabilities

On the left part of Figure 1 the DNN architecture is presented.

We cut the first 16384 symbols of the sequence. In case the sequence is shorter we pad it with zeros. Some sequences are reversed so we implement reverse augmentation for training. We take the last 16384, reverse them and replace A with T and C with G.

Firstly the model replaces each symbol of an input sequence to a vector of length 16 (Embedding layer). This means that the input sequence of length 16384 will be encoded as a 2D-matrix 16384×16 .

Then it multiplies a kernel matrix of size 16×35 to each position of the sequence matrix and results in a vector of $16384 - 35 + 1 = 16350$ values (Conv1D layer). There are 4096 kernels so we have a matrix 16350×4096 .

We are not interested in all these values. Each kernel describes a subsequence and we only want to know that the subsequence is located in the sequence. So we extract only max values that each kernel found (GlobalMaxPooling). If the max is big then the sequence contains the subsequence. As a result we have 4096 max values. The last layer is the lab-of-origin classifier.

On the right part of the image presented subsequences extractor. To prove that a particular sequence is classified correctly it is necessary to look inside the black box of DNN. For interpretation of the model we look at the output of GlobalMaxPooling values. If a value is bigger than 0 then it is an important value for the class (for simplicity in the description we suppose that all classifier weights are positive). Then we make one more step back and look at the matrix 16350×4096 . Find the position of max value that GlobalMaxPooling extracted for the particular kernel and define position (of length 35) in sequence that leads to positive values in the final vector. We attached a script that can define important parts of sequences based on a pretrained model and provide a set of subsequences which describe the lab-of-origin. In this script we consider that classifier weights can be negative.

Ensemble of several same architectures without plastic metadata gives top10 validation accuracy ~ 0.92 . With concatenated metadata it is ~ 0.95 .

Pros:

- Simple and interpretable model
- No any data preprocessing
- Fast inference (100+ sequences per second on Gpu)
- Simple and fast fine tuning for new labs

Cons:

- Top 10 accuracy of the model is not perfect (0.95 vs 0.97 for the best our model)

2.2 Big but more accurate model

Except the subsequences extraction we generated other features based on Blast output which were included to the final model.

Matrices of similarity between each DNA in train and test datasets.

The feature matrix is a square matrix where each row and column is sequence id and value on the intersection of sequence ids is the value that Blast provides for that pair. Blast provides a set of values for each found match: Identity, alignment length, number of mismatches, gap opens, query start, query end, subject start, subject end, evalue and bit

score. We found which values contain the most useful information and ended up with 3 different matrices based on $\log(\text{alignment length})$, mismatches and gap opens. The resulting matrices are too big for DNN input and we reduced them to 1024 main PCA components (but it's still possible to use them in raw form).

Example of BLAST similarity between two sequences:

query_id	9ZIMC_train
subject_id	LWFZU_train
identity	99.910
alignment length	3348
mismatches	1
gap opens	2
q.start	3805
q.end	7151
s.start	2117
s.end	5463
evalue	0.0
bit score	6165

Having such a table, we can construct a large square matrix of size $\text{len}(\text{train}) + \text{len}(\text{test}) = 63017 + 18816 = 81833$. See the figure.

The matrix is built without taking into account the target, so the test is safe to include in the vector and there are no problems with data leaks. In the matrix itself, 0 is put in position (i, j) - if, according to the results of launching Blast, it did not find a correspondence between the i and j sequences. If a match has been found, then you can put in this cell some kind of statistics $X_{i, j}$ that characterizes this coincidence. In our experiments, the following worked best (in descending order of quality):

- gap opens
- $\log_{10} p$ (alignment length)
- $\log_{10} p$ (bitscore)
- mismatches
- identity

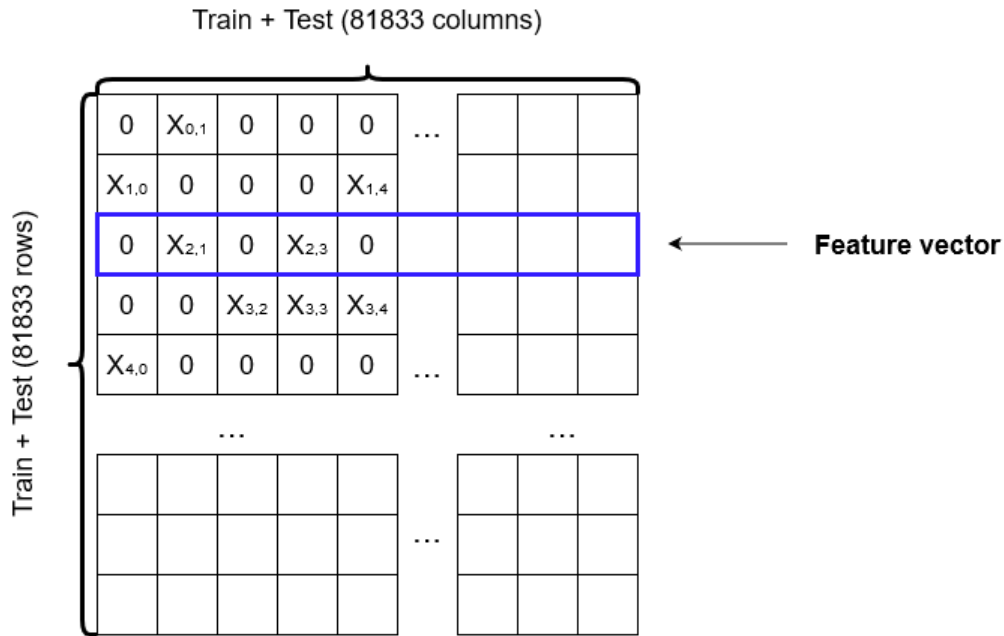


Figure 2. Feature vectors out of BLAST data

The vector data (size 81833) can be used to train the model for classification instead of sequences - which gives a similar result. When used in a model together with a sequence according to the scheme described in the previous section, the result is better, that is, there is synergy.

Since the size of 81833 is quite large and imposes memory constraints, it is better to use Principal Component Analysis (PCA) before launching to reduce vectors to, for example, 1024 components, or to store and use matrices in sparse form (since they have a large content of 0).

You can also increase the vector at the expense of other unlabeled data, that is, add not only test data, but also data from some other laboratories, which theoretically can improve the quality of the solution.

Blast class similarity matrix

From the Blast results, you can also get a matrix of a special kind, which can be used both as a preliminary answer to the problem (i.e. sent to the LeaderBoard), and as additional features for the neural network. Two such matrices were built, one for train (using Leave One Out scheme), and the second for test.

Form of the matrix is $(N \times 1314)$, where N is the number of sequences (in the case of this problem, (63017×1314) for train and (18816×1314) for test). When constructing the matrix, the target is used - therefore, it must be built according to the Out of Fold (OOF) scheme with dividing the training data into folds, or use the Leave One Out scheme as an extreme case.

When constructing the matrix, the data from the Blast program are used, which are built comparing all sequences (train + test) with all sequences (train + test).

How is the matrix built? We take a specific sequence, say it at position i in the train. Looking for all its matches with other sequences from the train sorted by bitscore. We look at the c/s class of the first match, and put the value 1 in position $(i, c/s)$. We look at the next match - if the class is the same, then skip and go to the next match, if the class is new $c/s1$, then we put the value $\frac{1}{2}$ in position $(i, c/s1)$, where 2 means the number of unique matching classes found so far. That is, for the next found class it will be $\frac{1}{3}$, for the next $\frac{1}{4}$, and for K class $\frac{1}{K}$.

If you send the matrix constructed this way to the LeaderBoard, you can immediately get a score of ~ 0.88 on Public. We used the matrices both for training and for mixing into the final ensemble.

The final architecture presented on Figure 3. The sequence analyzer is not changed but we added additional input data and concatenated all intermediate layers before final classification. Additional inputs are:

- Initial plasmids metadata without any changes. It contains unique information about the lab-of-origin so significantly improves metrics.
- 4 PCA from similarity matrices. Each has the same 2 layers architecture and concatenation with categorical features. This is not unique data because the information extracted with blast theoretically can be extracted from raw sequences with Embedding layer + Conv1D.

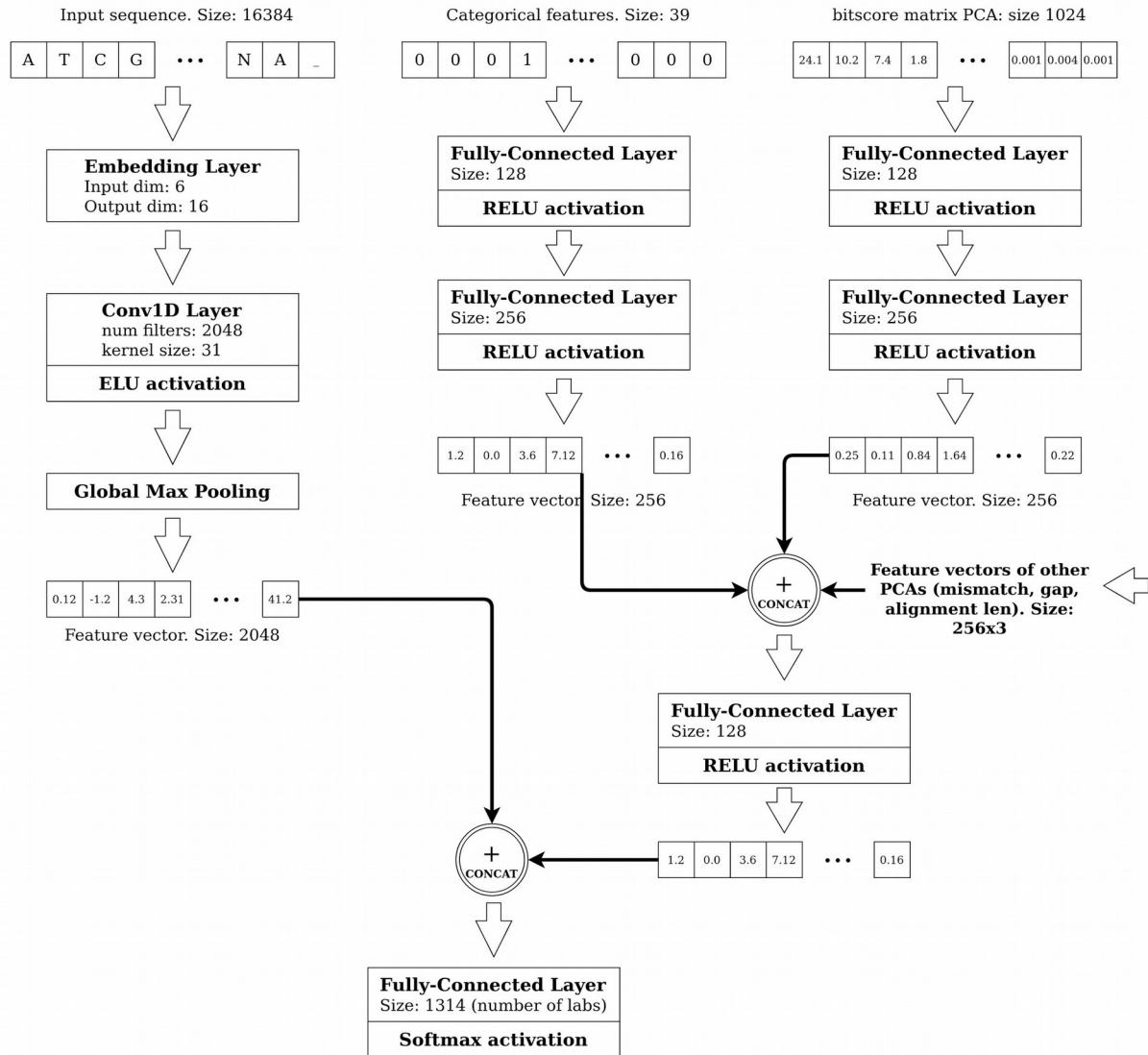


Figure 3: Our best model

Pros:

- Perfect top10 accuracy ~0.97 on validation.
- All data processed with a single model.

Cons:

- Requires complex preprocessing (launch Blast, build matrix, apply PCA)
- Difficult interpretation because of complex architecture

2.3 Top10 accuracy metric improvements

Our experiments helped to find peculiarities of post processing that improve top10 accuracy metric.

- Hard smoothing. Positive class = 0.7 instead of 1.0.
- Use logist layers instead of softmax layers for ensembling.
- Shift the prediction distribution to the test distribution.

Interesting thing is that these approaches do not often improve quality in other tasks and we believe that it works because of the specific metric. All these things together improve top10 accuracy on 0.01 while top1 accuracy almost not changed.

Ensemble

We create 9 different architectures with different inputs and different layer structure. It increases our score for around ~0.01 We made ensembles on normalized logits and at the end we normalize distribution to be the same as train which a little bit increased the score.

4) Copy and paste the 3 most impactful parts of your code and explain what each does and how it helped your model.

*** Code for our best model:**

```
def create_arch_v3(seq_kernels, left_padding_size, seq_size,
embs_channels_size):
    seqs_embeddings = []

    input_seqs = Input(shape=(left_padding_size + seq_size,))
    embs = Embedding(8, 24, mask_zero=True)(input_seqs)
    for i in seq_kernels:
        conv = Conv1D(embs_channels_size, i, activation='elu')(embs)
        conv = GlobalMaxPooling1D()(conv)
        conv = Dropout(0.01)(conv)
        seqs_embeddings.append(conv)

    input_cat = Input(39, name='input_cat')
    x_cat = input_cat
    x_cat = Dense(128, activation='relu', name='cat1')(x_cat)
    x_cat = BatchNormalization()(x_cat)
    x_cat = Dropout(0.15)(x_cat)
    x_cat = Dense(256, activation='relu', name='cat2')(x_cat)
    x_cat = BatchNormalization()(x_cat)
    x_cat = Dropout(0.15)(x_cat)

    input_blast = Input(1314, name='input_blast')
    x_blast = input_blast
    x_blast = Dropout(0.4)(x_blast)
    x_blast = Dense(256, activation='relu')(x_blast)
    x_blast = BatchNormalization()(x_blast)
    x_blast = Dropout(0.1)(x_blast)

    input_gap = Input(1024, name='input_gap')
    x_gap = input_gap
    x_gap = Dense(512, activation='relu')(x_gap)
    x_gap = BatchNormalization()(x_gap)
```



```

x_gap = Dropout(0.5)(x_gap)
x_gap = Dense(128, activation='relu')(x_gap)
x_gap = BatchNormalization()(x_gap)
x_gap = Dropout(0.5)(x_gap)

input_bitscore = Input(1024, name='input_bitscore')
x_bitscore = input_bitscore
x_bitscore = Dense(512, activation='relu')(x_bitscore)
x_bitscore = BatchNormalization()(x_bitscore)
x_bitscore = Dropout(0.5)(x_bitscore)
x_bitscore = Dense(128, activation='relu')(x_bitscore)
x_bitscore = BatchNormalization()(x_bitscore)
x_bitscore = Dropout(0.5)(x_bitscore)

input_mismatch = Input(1024, name='input_mismatch')
x_mismatch = input_mismatch
x_mismatch = Dense(512, activation='relu')(x_mismatch)
x_mismatch = BatchNormalization()(x_mismatch)
x_mismatch = Dropout(0.5)(x_mismatch)
x_mismatch = Dense(128, activation='relu')(x_mismatch)
x_mismatch = BatchNormalization()(x_mismatch)
x_mismatch = Dropout(0.5)(x_mismatch)

input_alen = Input(1024, name='input_alen')
x_alen = input_alen
x_alen = Dense(512, activation='relu')(x_alen)
x_alen = BatchNormalization()(x_alen)
x_alen = Dropout(0.5)(x_alen)
x_alen = Dense(128, activation='relu')(x_alen)
x_alen = BatchNormalization()(x_alen)
x_alen = Dropout(0.5)(x_alen)

x_pcas = Concatenate(axis=1)([x_alen, x_cat, x_blast, x_gap,
x_bitscore, x_mismatch])
x_pcas = Dense(2048, activation='relu')(x_pcas)
x_pcas = Dropout(0.2)(x_pcas)

seqs_embeddings.append(x_pcas)
x = Concatenate(axis=1)(seqs_embeddings)

x = Dense(1314, name='final_dense')(x)

x = layers.Softmax(dtype='float32')(x)
return Model([input_seqs, input_cat, input_blast, input_alen,
input_gap, input_bitscore, input_mismatch], x)

```

Model has 7 different inputs - raw sequence limited to size=seq_size, 39 categorical features, BLAST predicted classes, 3 feature vectors from BLAST matrices based on gap, bitscore, mismatches and alignment length. Output is 1314 vector with softmax output.

*** Code to extract PCA from large BLAST matrix:**

```
def create_pca_for_single_matrix(file_path, transformer_path):
    from sklearn.decomposition import PCA
    n_components = 1024
    print(file_path)
    ids1, matrix = load_from_file(file_path)
    print('Matrix read complete... {} {}'.format(len(ids1),
matrix.shape))
    print('Create PCA...')
    transformer = PCA(n_components=n_components, random_state=0)
    transformer.fit(matrix)
    save_in_file_fast(transformer, transformer_path)
    X = transformer.transform(matrix)
    print('Transform complete!')
    ids = [x.split('_')[0] for x in ids1 if 'train' in x]
    PCA = pd.DataFrame(data=X[:len(ids)], columns=['f' + str(i)
for i in range(1024)], index=ids)
    PCA.index.name = 'sequence_id'
    PCA.to_csv(FEATURES_PATH + os.path.basename(file_path)[:5] + '.csv')
```

This code takes on input large 81833x81833 matrix and returns 81833x1024 matrix with most impactful features.

*** Code to weighted ensemble of predictions:**

```
def ensemble_submissions(subm_list, mode):
    CLASSES = get_classes()

    seq_check = None
    matrix_full = None
    real_data = pd.read_csv(INPUT_PATH +
'{}_values.csv'.format(mode), usecols=['sequence_id'])
    seq_check = tuple(real_data['sequence_id'])

    weights = []
    for path, w in subm_list:
        weights.append(w)
    print('Weights:', list(weights))

    weights = []
    for path, w in subm_list:
        weights.append(w)
    print('Read {} Weight: {}'.format(path, w))
```

```

s = pd.read_csv(path)
s = real_data[['sequence_id']].merge(s, on='sequence_id',
how='left')
s.fillna(0, inplace=True)
if not seq_check:
    seq_check = tuple(s['sequence_id'])
else:
    if seq_check != tuple(s['sequence_id']):
        print('Some error here!')
matrix = s[CLASSES].values
if NORMALIZE_V2:
    if matrix.max() != 1:
        print('Normalize!')
        matrix = normalize_v2(matrix).astype(np.float32)
    else:
        if USE_SQRT:
            print('Sqrt!')
            matrix = np.sqrt(matrix)

if matrix_full is None:
    matrix_full = matrix.astype(np.float32).copy() * w
else:
    matrix_full += matrix.astype(np.float32).copy() * w
print('Weights:', list(weights))
matrix_full /= np.array(weights, dtype=np.float32).sum()
s[CLASSES] = matrix_full
if mode == 'holdout':
    subm_path = SUBM_PATH + 'submission.csv'
else:
    subm_path = SUBM_PATH + 'submission_{}.csv'.format(mode)
print('Final submission located at: {}'.format(subm_path))
s.to_csv(subm_path, index=False)

if USE_FINAL_DISTRIBUTION_NORM:
    norm_subm_path = subm_path[:-4] + '_normalized.csv'
    s = normalize_submission_distribution_v1(s)
    s.to_csv(norm_subm_path, index=False)
    print('Normalized submission located at:
{}'.format(norm_subm_path))

return subm_path

```

This code takes on input set of submissions with weights and ensemble them to get final predictions.

5) Please provide the machine specs and time you used to run your model.

- CPU (model): AMD Ryzen Threadripper 2920X 12-Core Processor
- GPU (model or N/A): 4 x NVIDIA 1080 Ti 11 GB
- Memory (GB): 128 GB
- OS: Windows 10
- Train duration: ~144 hours (on single GPU). We have 9 different neural net models so on 4 GPUs system it is around 36 hours total.
- Inference duration: Around 1 hour for competition test set on single GPU (excluding BLAST run time).

6) What are some other things you tried that didn't necessarily make it into the final workflow (quick overview)?

We find out that the simplest model which gives good results for this problem is following:

```
inp = Input((16384, 6))
x = Conv1D(2048, 31, activation='elu')(inp)
x = GlobalMaxPooling1D()(x)
x = Dense(out_channels, activation='softmax')(x)
model = Model(inputs=inp, outputs=x)
```

It uses only sequences and after training gives around ~0.916 TOP10 Acc on validation.

7) Did you use any tools for data preparation or exploratory data analysis that aren't listed in your code submission?

No, we only used BLAST.

8) How did you evaluate performance of the model other than the provided metric, if at all?

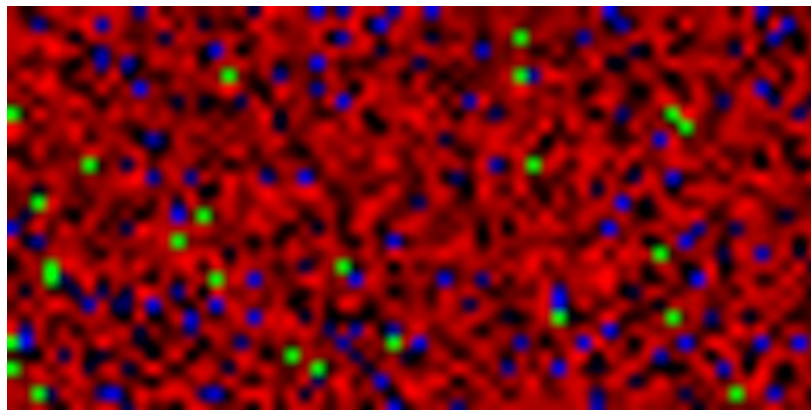
We used the TOP10 accuracy metric only.

9) Anything we should watch out for or be aware of in using your model (e.g. code quirks, memory requirements, numerical stability issues, etc.)?

There can be some small fluctuation with PCA results - they are saved as CSV and eventually last digits in numbers a little bit different on different systems. But overall results don't affect that much. Almost all TOP10 entries for predictions stay the same.

10) Do you have any useful charts, graphs, or visualizations from the process?

We tried to visualize activation maps for the Conv1D layers, but it wasn't really useful.



11) If you were to continue working on this problem for the next year, what methods or techniques might you try in order to build on your work so far? Are there other fields or features you felt would have been very helpful to have?

There are set of methods that can be tried:

- 1) Metric Learning - an approach in which each sequence will be associated with a vector of fixed length. All vectors of one laboratory will lie close to each other (according to the Cosine Sim metric). Then, by comparing the vector of the new sequence with all other vectors, it will be possible to select the cluster of vectors that lies close to each other. The PROs of this method that you can add new labs without retraining the actual model.
- 2) Siamese nets for a direct comparison of two sequences. At the output, we get 0 - if different laboratories, and 1 - if the same laboratory. It's similar to metric learning.
- 3) Use transformers. First, SentencePiece on all sequences from Train to get a dictionary, then we tokenize the sequences and send them to the network input (for example, BERT).

4) We can make spectrograms on sequences and use neural networks for image classification (EfficientNet or DenseNet).