1st Place Winning Solution - Mechanisms of Action (MoA) Prediction

Overview

Thanks to the Kaggle team and Laboratory for Innovation Science at Harvard who hosted this challenging and interesting MoA competition!

Representing the *Hungry for Gold* team, in this post I'm going to explain our winning solution in detail.

Our winning blend consists of 7 single models:

- 3-stage NN stacking by non-scored and scored meta-features
- 2-stage NN+TabNet stacking by non-scored meta-features
- SimpleNN with old CV
- SimpleNN with new CV
- 2-heads ResNet
- DeepInsight EfficientNet B3 NS
- DeepInsight ResNeSt

The following overview diagram depicts our winning blend models.

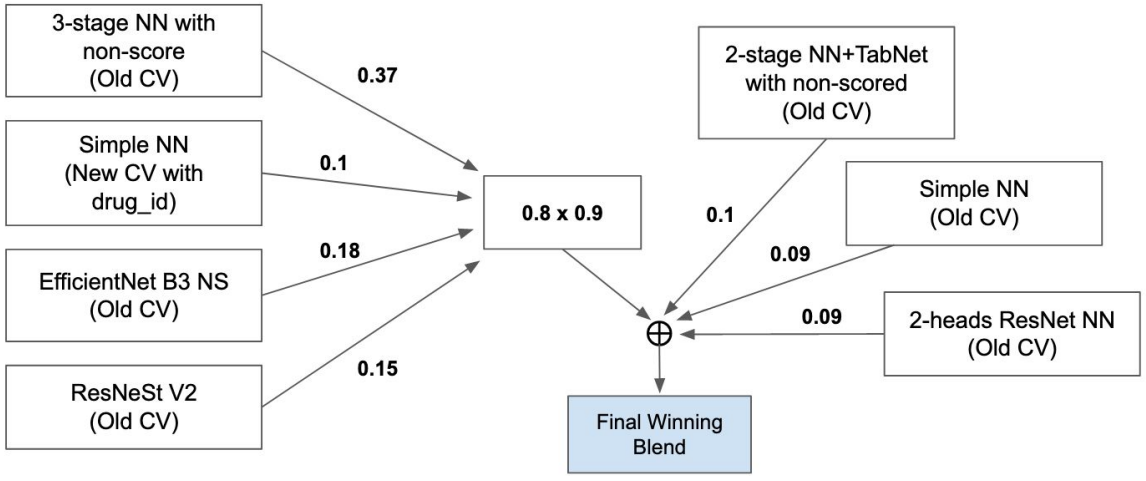

Figure 1. Winning Weighted-Average Blend.

Our two final submissions are based on weighted average. The winning blend is the best LB (Table 1), and the best CV blend (Table 3) can also achieve 5th place in the private LB. It shows that both final submissions are performant and robust.

Table 1. Winning Blend with 7 Models (Private LB: 0.01599).

Single Model	Trained with control group	Seeds/ K-folds	Old CV	New CV	Public LB	Private LB
3-stage NN	No	5/5	0.01561		0.01823	**0.01618**
2-stage NN+TabNet	No	5/10	0.01615		0.01837	0.01625
Simple NN	No	6/5	0.01585		0.01833	0.01626
	No	6/5		0.01564	0.01830	**0.01622**
2-heads ResNet	No	7/10	0.01589		0.01836	0.01624
EfficientNet B3 Noisy Student	Yes	1/10	0.01602		0.01850	0.01634
ResNeSt V2	Yes	1/10	0.01576		0.01854	0.01636

For our winning blend, the selection of blend weights were based on two factors: the LB scores and correlations between the single models. For better understanding, the mean correlation of each model's predictions without control groups was calculated with respect to other single models (shown in Table 2). The selection was manually done to maximize the LB score. Models with less mean correlations were given higher weights. Also, some of the weights were decided by the previous blend submission scores. More details can be found in the Model Diversity section.

Table 2. Mean Correlation Between the Submissions of Best Single Models.

Single Model	Mean Correlation with other Single Models	Public LB
3-stage NN	**0.77**	**0.01823**
2-stage NN+TabNet	**0.77**	0.01837
Simple NN (old CV)	0.81	0.01833
Simple NN (new CV)	0.82	0.01830
2-heads ResNet	0.80	0.01836
EfficientNet B3 Noisy Student	**0.74**	0.01850
ResNeSt V2	**0.70**	0.01854

The selection of single models in our Best CV blend is fully based on the OOF (Out-of-folds) predictions. We used the TPE (Tree-structured Parzen Estimator) sampler in Optuna and SLSQP (Sequential Least Squares Programming) method in Scipy for searching CV-optimized weights. The resulting weights from Optuna (under 3000 or 5000 trials) and SLSQP are nearly the same.

The log loss of the best CV is **0.15107 (private LB: 0.01601)** from 5 models. Interestingly, the search results eliminated 2-stage NN+TabNet and Simple NN models, which gave some contribution to the winning blend.

Table 3. Best CV Blend with 5 Models (CV: 0.015107, Private LB: 0.01601).

Single Model	Weight	Trained with control group	Seeds/ K-folds	Old CV	Public LB	Private LB
3-stage NN	0.37	No	5/10	0.01546	0.01822	**0.01618**
2-heads ResNet V2	0.08	No	7/10	0.01566	0.01844	**0.01623**
EfficientNet B3 Noisy Student	0.17	Yes	1/10	0.01602	0.01850	0.01634
ResNeSt V1	0.11	Yes	1/10	0.01582	0.01853	0.01636
ResNeSt V2	0.27	Yes	1/10	0.01576	0.01854	0.01636

The addition of DeepInsight CNNs played a significant role in our final blends because of their high diversity to other shallow NN models. By curiosity, we compared the log loss scores of our winning blend with/without those CNN models (shown in Table 4).

We got almost **0.00008** improvement in the private LB score while including DeepInsight CNN models!

Table 4. Winning Blends with/without DeepInsight CNNs.

Blend	Public LB	Private LB
NN-based + TabNet Only	0.01813	0.01607
NN-based + TabNet + DeepInsight CNNs	0.01808 **(-0.00005)**	0.01599 **(-0.00008)**

Cross-Validation Strategies

Most of our models are based on the MultilabelStratifiedKFold (old CV), with the exception of a Simple NN model that used the new double stratified CV shared by Chris using drug_id information. We used different seeds for the CV splits in our models. The choice of CV strategy is based on the goodness of alignment to the CV log loss and LB log loss. The *K* of CV is either 5 or 10. We also trained the models using multiple seeds to reduce the variance.

Notably, we noticed that it is difficult to combine the OOF files from both old and new CVs for searching optimized CV weights as the log loss scores of new CV models are much higher. We ended up selecting only old CV models for our Best CV blend, which scored **0.15107** on old CV and **0.01601** on private LB.

Model Details

In our teamwork, Nischay and Kibuna contributed on feature engineering and multi-stage stacking of 3-stage NN, 2-stage NN+TabNet and Simple NN models while I was focusing on DeepInsight CNNs and the replication of a 2-heads ResNet model.

To fight with the risk of overfitting under this small and highly imbalanced multi-label dataset, we imposed label smoothing and weight decay as the regularization methods in the model training process. Label smoothing worked very well and prevented our models from being too confident about its predictions, which greatly reduced the chance of overfitting.

To make our models more visually understandable, we have provided a model architecture diagram for each of our single models to explain the inner training process and its NN topology. In this section we are going to introduce each model in detail.

3-Stage NN

This is our best single model (CV: **0.01561**, Public LB: **0.01823**, Private LB: **0.01618**) and a great addition to our final blend on both CV and LB scores. Nischy and Kibuna did a great job on producing good results based on the idea of multi-stage stacking with model predictions as meta-features and different engineering features. Control group rows are removed from the training set.

Model Architecture

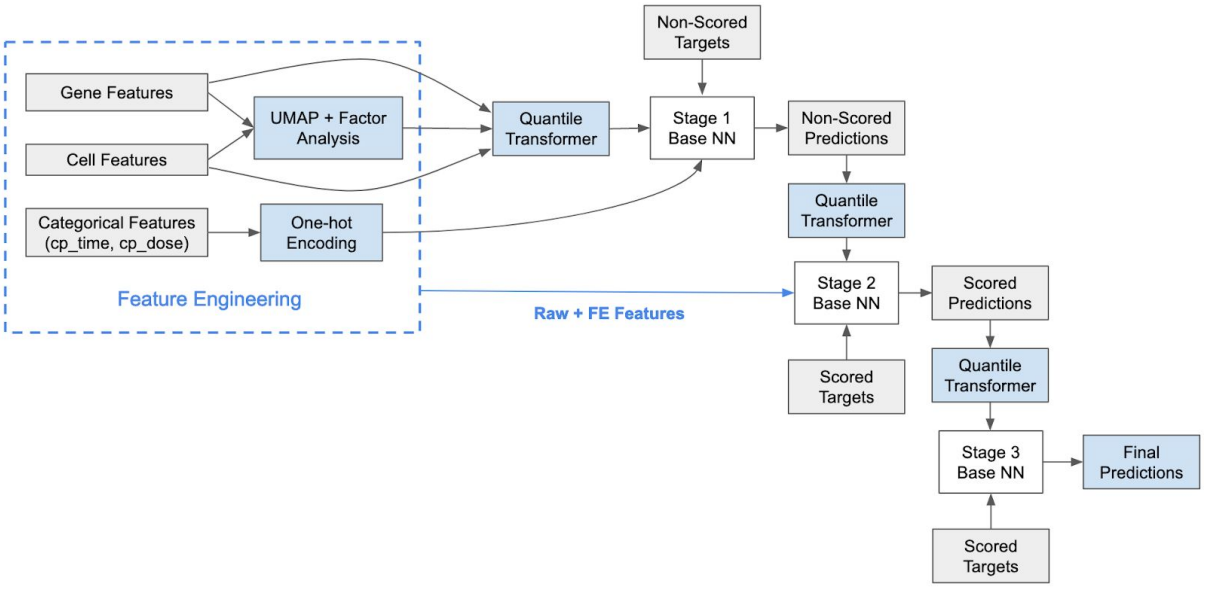

Figure 2. 3-stage NN Model.

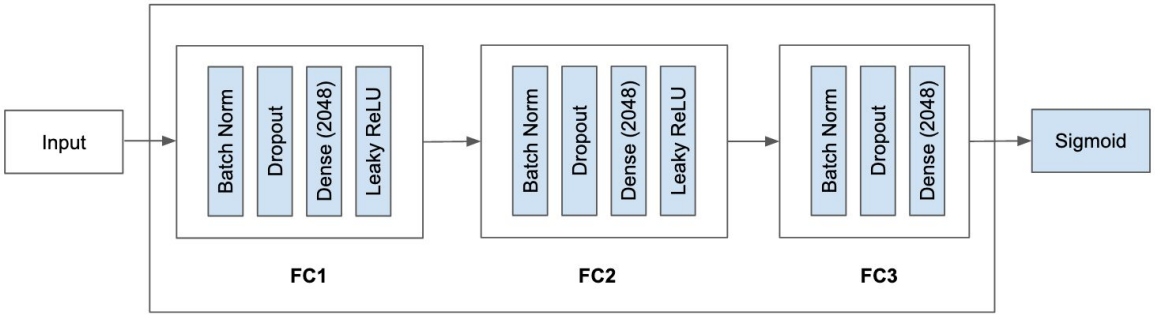

Figure 3. 3-FC Base NN Block Topology.

Figure 3 depicts the detailed topology of the model's based NN block, which is similar to the public notebooks. Some changes in the training setup were made to boost up the performance. In the multiple stages training, the same architecture was used apart from tuned dropout values.

STAGE 1

umap似乎本身就常用于细胞数据的降维

Firstly, UMAP and Factor Analysis were used to generate extra features from the genes and cells features. The Quantile Transformer was applied to all features except for one-hot encoding ones. In this stage the NN model with a size of 2048 hidden neurons is trained for 15 epochs for non-scored targets excluding those having zero values (332 targets included). The non-scored predictions were reused as meta-features for the next stage.

STAGE 2

In the 2nd stage, we applied Quantile Transformer again on the non-scored meta-features to combine with the original features in the previous stage, and trained another NN for 25 epochs with a size of 2048 hidden neurons for scored targets. Similarly, the scored predictions were reused as meta-features for the next stage.

STAGE 3

In the last stage, we also applied Quantile Transformer on the scored meta-features, and retrained a NN for 25 epochs with a size of 1024 hidden neurons based on only those meta-features, while targets were clipped from 0.0005 and 0.9995 in the training data after label smoothing.

In each stage a similar setup was used: a Adam optimizer with a learning rate of 5e-3, a batch size of 256, a weight decay of 1e-5, and a OneCycleLR scheduler with a maximum learning rate of 1e-2.

The final model produced a CV log loss of **0.01561**, which is a bit higher than the 2nd stage model due to clipping to model predictions, but it scored really well on the LB.

2-Stage NN+TabNet

Similar to the 3-stage NN, this model also used non-scored and scored targets in multi-stage training. TabNet is only used for the 2nd stage instead of the first one, because it was not producing good results on non-scored targets. Control group rows are removed from the training set.

Figure 4 and 5 show the model architecture and the detail of Base NN used in the first stage.

Model Architecture

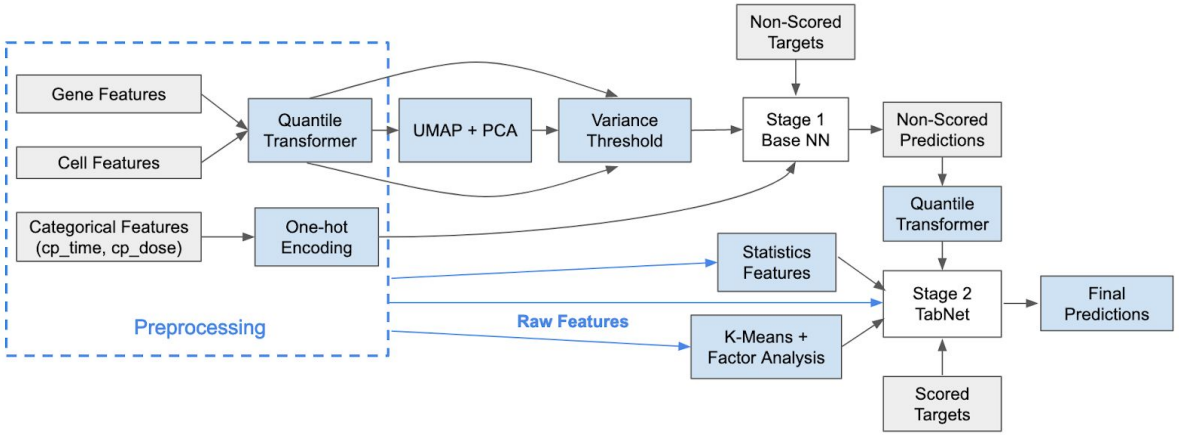

Figure 4. 2-stage NN+TabNet Model.

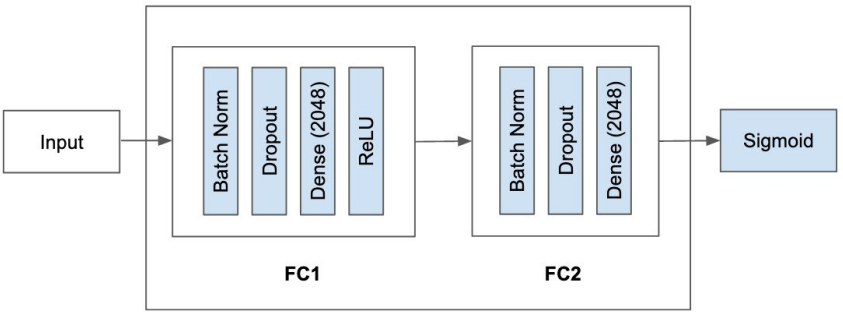

Figure 5. 2-FC Base NN Block Topology.

Stage 1

For the feature engineering part, PCA and UMAP were used along with QuantileTransformer on the genes and cells features, with a variance threshold of 0.5 to keep only important features. A NN model with a size of 2048 hidden neurons is trained for 15 epochs for non-scored targets. Those predictions are then used as meta-features for the next stage.

The following training setup was used: a Adam optimizer with a learning rate of 5e-4, a batch size of 256, a weight decay of 1e-5, and a OneCycleLR scheduler with a maximum learning rate of 1e-2.

Stage 2

In the 2nd stage, we applied Quantile Transformer again on the non-scored meta-features to combine with the original features in the previous stage, and trained a TabNet with label smoothing (0.0001) for scored targets. Factor Analysis, K-Means Clustering along with some statistics features like mean, skew, kurtosis, and std of genes and cells features were also used.

PyTorch TabNet regressor training setup: a width of 32 for the decision prediction layer, and a width of 32 for the attention embedding for each mask, 1 step in the architecture, a gamma value of 0.9, Adam optimizer with a learning rate of 2e-2 and a weight decay of 1e-5, a sparsity loss coefficient of 0, and entmax as the masking function. It was trained with a batch size of 1024 and a virtual batch size of 128 for 200 epochs before early-stopped by a patience of 50 epochs. The final TabNet model produced a CV log loss of **0.01615**.

Simple NN

The model topology of simple NN is the same as the 3-FC Base NN used in the 3-stage NN model. It is trained with only a single stage on scored targets. Figure 6 and 7 show the model architecture and the detail of Base NN. Control group rows are removed from the training set.

Model Architecture

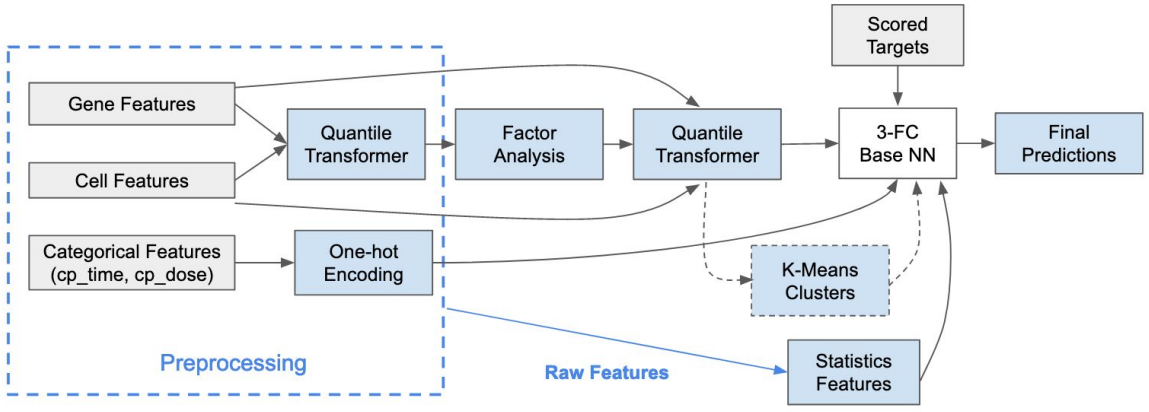

Figure 6. Simple NN Model.

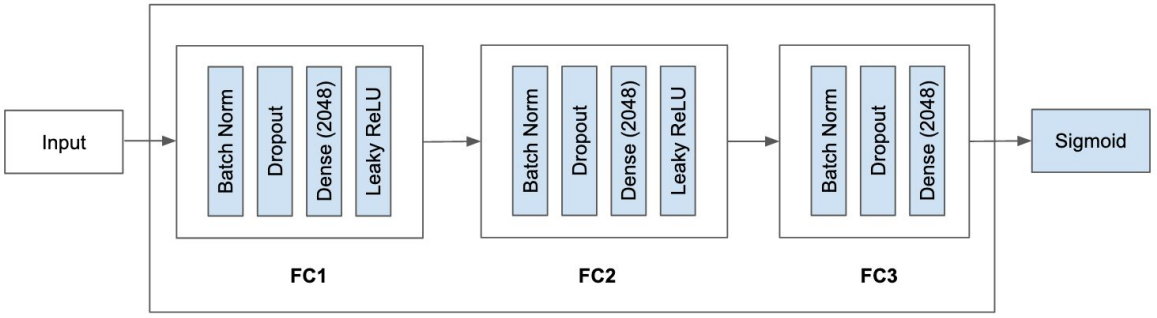

Figure 7. 3-FC Base NN Block Topology.

For the feature engineering part, Factor Analysis is used along with QuantileTransformer on the genes and cells features except for one-hot encoding ones. Some statistics features like mean, skew, kurtosis, and std of genes and cells features were also used. A NN model with a size of 2048 hidden neurons is trained for scored targets.

The following training setup was used: a Adam optimizer with a learning rate of 5e-4, a batch size of 256, a weight decay of 1e-5, an early stopping patience of 10 epochs, and a OneCycleLR scheduler with a maximum learning rate of 1e-2.

We trained two Simple NN models in total, one with the old CV including extra features from K-Means Clustering, and the other one with the new CV to add more diversity.

2-heads ResNet

This model replicated the public notebook "Fork of 2heads looper super puper plate" shared by Demetry Pascal with some minor updates for the freezing/unfreezing training loops. The original notebook was written in R (public LB: **0.01833**, private LB: **0.01624**), and we made a Tensorflow/Keras Python version for it (public LB: **0.01836**, private LB: **0.01624**). Control group rows are removed from the training set.

Figure 8 and 9 show the model architecture and the detail of ResNet-like Residual NN.

Model Architecture

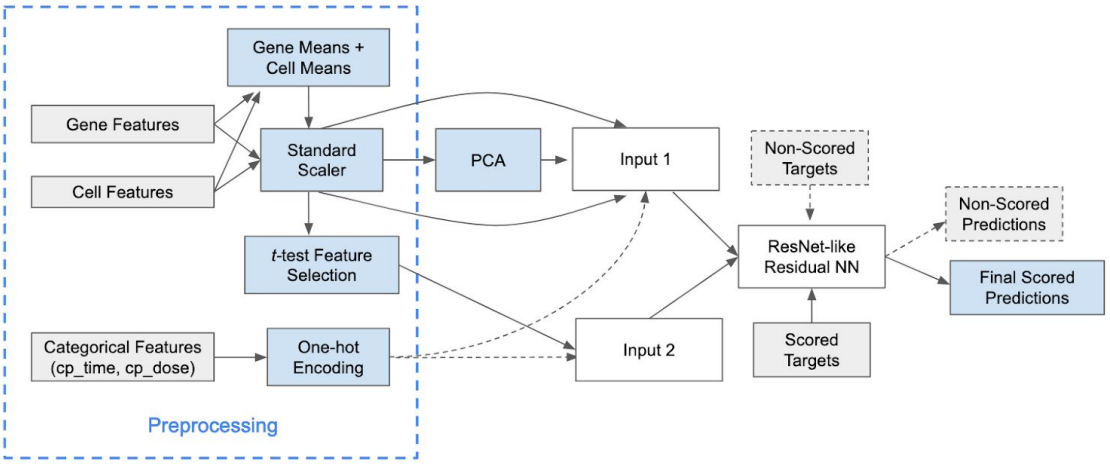

Figure 8. 2-heads ResNet Model.

The architecture is based on ResNet-like Residual NN with two inputs, one is from mean and PCA components of genes and cells features, and the other one includes the 447 selected important features by Student's t-test based on the 25% quantile of the p-values for the test of sample means between training rows with and without MoA. All features are normalized by Standard Scaler.

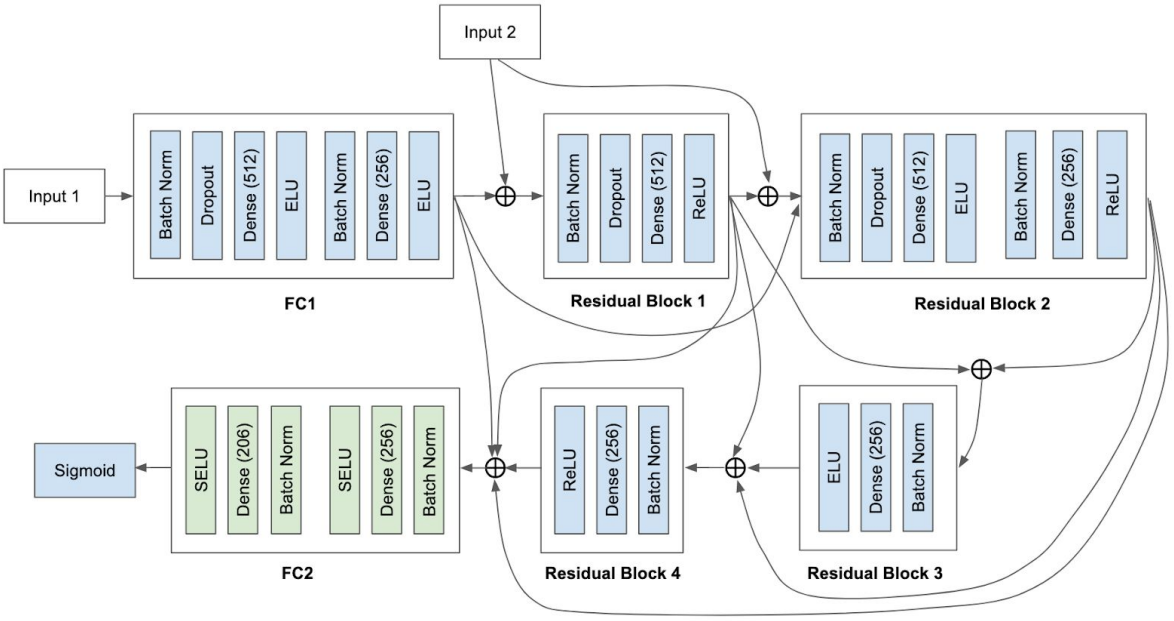

Figure 9. ResNet-like Residual NN Topology.

Using a ResNet-like topology brings the benefit of skip connections in the FC layers, which allows the flow of information to be passed from one layer to next layers. In this way, the model gains better capability of learning with more hidden neurons and layers without the early degradation problem.

The training process can be divided into two phases.

Firstly, the model is trained for non-scored targets by Adam optimizer for 50 epochs minimizing the binary cross entropy loss, with a learning rate of 1e-3, a batch size of 128, early stopping with a patience of 10 epochs and a minimum delta of 1e-5. ReduceLROnPlateau learning rate scheduler is used with a factor of 0.5 and a patience of 4 epochs. The alpha value of label smoothing is set to 0.0005.

In the second phase, the learnt model weights from non-scored targets are being reused and transferred learning to scored targets using loops of freezing and unfreezing process. The weights except for the last layers (the FC2 module in Figure 9) are freezed for training several epochs with minimum delta of early stopping set to 1e-6, and a smaller learning rate of 1e-3/3. And then the whole model weights are unfreezed and we follow a similar loop to train the same model, with an even smaller learning rate of 1e-3/5, until it early-stopped.

There is also another V2 version of model that included one-hot encoding features and used a smaller batch size of 64. It got a worse public LB but scored better on CV and private LB, which contributed as part of our best CV blend.

DeepInsight CNNs

Compared with most of the shallow NN models shared in public notebooks and our other single models that added lots of engineering features, our DeepInsight CNNs only used raw features from genes and cells!

The idea was to fully utilize the power of feature extraction and transfer learning from pretrained convolutional neural networks and generate highly unique and diverse models in comparison with other NN models.

As introduced in my post about DeepInsight, instead of doing feature extraction and selection for collected samples (*N* samples x *d* features), we are **arranging similar or correlated features into the neighboring regions of a 2-dimensional feature map (*d* features x *N* samples)** to ease the learning of their complex relationships and interactions.

Since we are transforming features instead of samples, the shape of the learnt feature map is static for samples. However, the distribution of features values (heatmap colors) in the feature map per class looks different. As we know, CNNs can learn hierarchical patterns and exploit spatially local correlation by enforcing a sparse local connectivity pattern between neurons of adjacent layers. Even if the feature map is fixed, the variance of feature values and the interactions of neighboring weak features are still the learnable source for kernel weights.

In this competition, we applied DeepInsight with two popular pretrained CNN models: EfficientNet and ResNeSt (Split-Attention Networks). And it turns out that they all worked very well on MoA data!

Figure 10 and 11 show the detail of DeepInsight CNNs.

Model Architecture

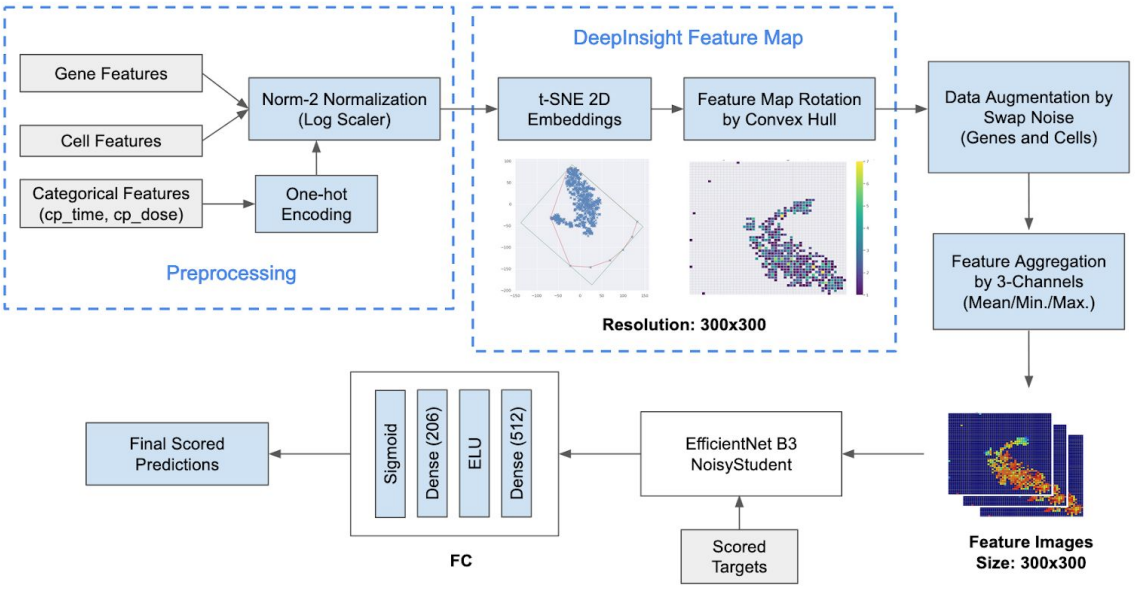

Figure 10. DeepInsight EfficientNet B3 NS Model.

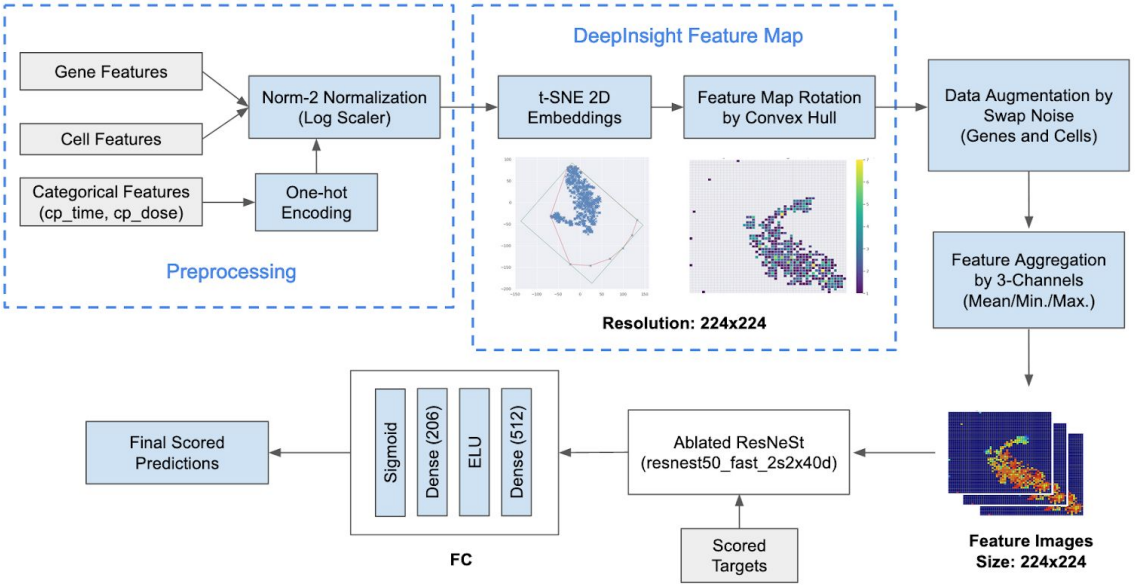

Figure 11. DeepInsight ResNeSt Model.

Training Process

The overall training process is the same for both CNN models. First of all, raw genes, cells and one-hot encoding features are normalized by the Norm-2 Normalization (Log Scaler) method mentioned in the original DeepInsight paper, which works better for CNNs than Standard Scaler, Robust Scaler and Min/Max Scaler in our experiments.

Secondly, t-SNE is applied to transform raw features into a non-linear 2D embeddings feature space. Then the convex hull algorithm is used to find the smallest rectangle

containing all features and a rotation is performed to align the frame into a horizontal or vertical form. After this step, we have created a **DeepInsight Feature Map** that contains the extracted neighboring relationships between the raw features from genes and cells.

Next, to deal with the chance of overfitting in this small and highly imbalanced dataset, we borrowed the idea of Swap Noise for data augmentation, where each training sample has a 10% chance of swapping 15% of the features with the other random sample in the training set. Our experimental results showed that this trick greatly reduced the chance of overfitting and boosted the prediction performance of CNNs.

Fourthly, the augmented raw feature values are mapped into the pixel coordinate locations of the feature map image. Note that the resolution of feature map image affects the ratio of feature overlaps (the features mapped to the same location are averaged), which is a trade-off between the level of lossy compression and computing resource requirements (e.g., host/GPU memory, storage).

In the original paper overlapped feature values are aggregated by the mean, and it provided only a single gray-scale channel duplicated into 3-channels for pretrained CNNs. Therefore, in the final step I thought of an idea of extracting also the minimum and maximum values as the other two channels to keep more statistical information about the overlapped values. This trick has been proven to boost the prediction performance on both CV (**-0.00013**) and public LB (**-0.00011**) in the experiment results significantly.

We chose to use the same value for resolution and image size, which is 300x300 for EfficientNet B3 NoisyStudent and 224x224 for ablated ResNeSt. The last FC layer of CNNs are replaced with a new FC with a hidden size of 512 and the ELU activation.

For ResNeSt, we chose the ablated version (***resnest50_fast_2s2x40d***), which provides nearly the best prediction performance of ResNeSt variants with relatively smaller model size and faster inference speed. We also trained another V2 model with a different seed and a dropout rate of 0.2 to the final FC layer.

Both CNNs are using the same training process implemented by the nice PyTorch Lightning framework. We trained the CNNs under 16-bit precision for about 35 epochs in 10-folds by the RAdam optimizer, with a learning rate of 0.000352 obtained from Learning Rate Finder, a batch size of 48 for EfficientNet B3 NS and 128 for ablated ResNet, a patience of 12 epochs for early stopping, a Cosine Annealing LR scheduler with T_max set to 5 epochs, and a label smoothing of 0.001 for binary cross entropy loss.

Each of them took about 12-25 hours to train for 10-folds. Therefore, they were all trained on my local machine (with two 2080-Ti GPUs) instead of a Kaggle GPU notebook, which has a run time limit of 9 hours.

In contrast to other models that removed control group rows from the MoA training set before training, I was having a different thought about it. From a discussion thread in the competition forum, the host was mentioning that "*... However, in real applications, the null controls are one way to assess the efficacy of a drug and can be useful in modeling the other MoAs*." This hint lets me think that control rows might provide some useful information

for the CNNs as they are modeling the relationships between gene expression and cell viability features. Therefore in our CNNs they are all trained with control group rows. Removing those rows produced worse scores on both CV and LB in the experiments.

Things that I had experimented but did not work:

- EfficientNet B0, B5, B7 NS (smaller models are weaker and bigger models overfitted more quickly in this small dataset)
- PCA or Kernel-PCA with Norm-2 Normalization, Quantile Transformer or Standard Scaler
- Higher dropouts in the last FC layers
- ImageNet normalized RGB channels
- Cosine annealing with warm restarts
- Other activation functions like ReLU, SELU, GELU, etc. in the last FC layers

Model Diversity

To understand more about the diversity of our models, here are the mean correlation heatmaps of our best single models. The correlation coefficients are calculated by Pearson's method for each pair of models. We took the mean of target-wise correlation coefficients as the final value of each model.

For our best LB model submissions, even though they all scored very well on LB, their correlations are impressively low! Especially for DeepInsight CNNs, the correlations are only around **0.63-0.73** with other NN models. It showed a strong proof of high diversity and a great source of ensemble boost.

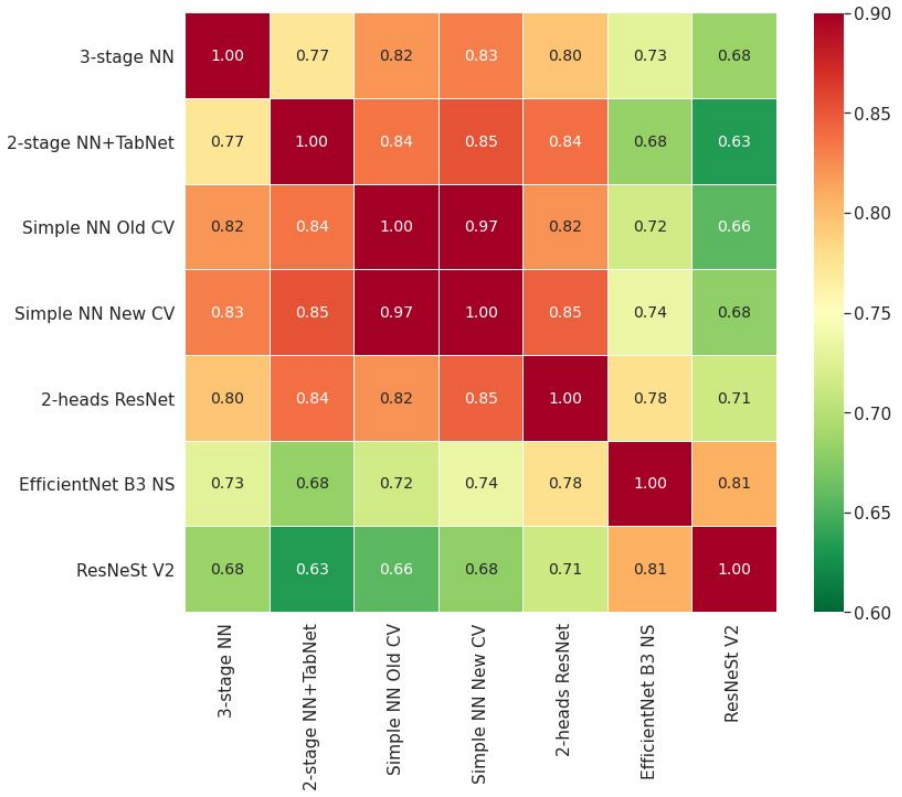

Figure 12. Mean Correlation Heatmap of Best LB Models' Submission Predictions.

The OOF (out-of-folds) predictions of our Best CV models also showed a similar phenomenon, and model correlations are even lower, with a coefficient of only around **0.52-0.70**!

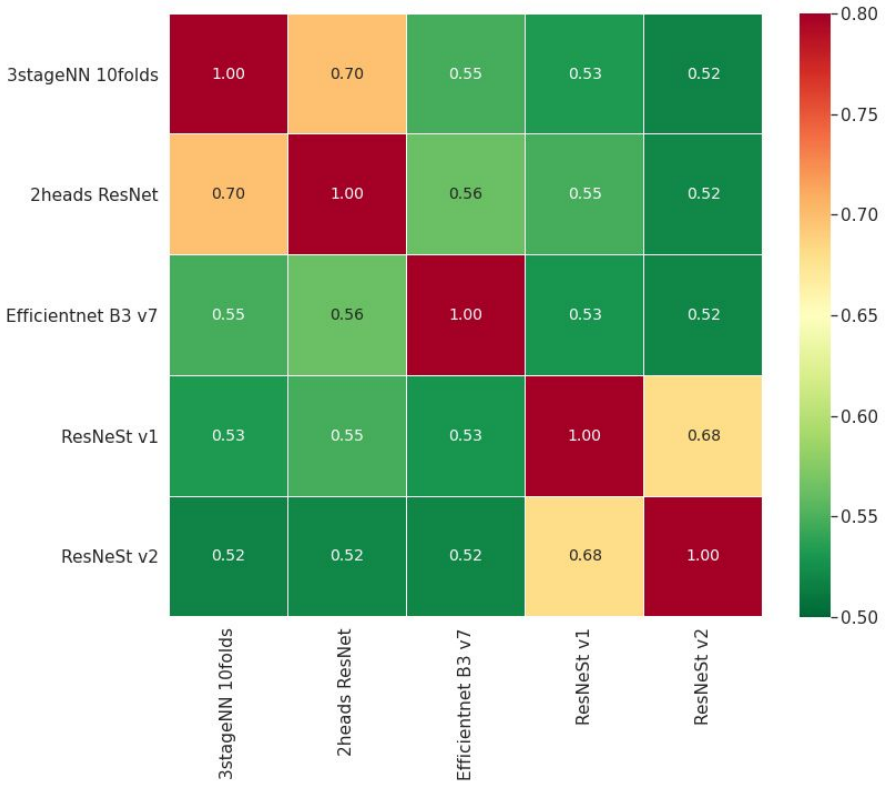

Figure 13. Mean Correlation Heatmap of Best CV Models' OOF Predictions.

In my previous experience, diverse models with correlations around **0.85-0.95** are usually very good sources for the ensembles. In this competition our best single models had proven their strong prediction performance and at the same time with much lower correlations than usual!

Inference Efficiency

To ensure that we can safely squeeze all our best models in the final blend submissions in 2-hours execution limit, we optimized the inference scripts by loading all scalers and fitted transformers from pickle files. Also, since the computing resource of Kaggle GPU notebook is rather limited (2-cores CPU, 13GB Host RAM and 16GB GPU RAM), we did several trials to find the sweet points for each model to maximize the inference throughput by the number of workers and batch size.

Regarding the batch size, our best setup is 2048 for NN-based models and 512 for CNN-based models (since CNN models are bigger and memory hungry). Setting bigger numbers won't help with the inference speed due to the saturation of CPU, GPU and I/O resources.

The total run time of final submissions are 1923 seconds (Best LB) and 1829 seconds (Best CV) on the public test set respectively, which all finished in 2 hours on the private test set.

Source Code

All of our training notebooks, blend notebooks and inference scripts are open-sourced in the following Github repository:
https://github.com/guitarmind/kaggle_moa_winner_hungry_for_gold

Note that the structure of this repository will be updated in the next few days to follow the *Winning Model Documentation Guidelines* by the Kaggle team.

The source code of our final submission models for training and inference can be found under the “final” folder.

Conclusion

Without any prior knowledge about MoA or the medical domain, we have done our best to apply the ML/DL techniques that we know to this small and highly imbalanced dataset. We are pretty amazed by our final standing, and it turns out that general AI methods could work in almost any domain.

The combination of shallow NN models in different architectures, in-depth feature engineering, multi-stage stacking, label smoothing, transfer learning, and the addition of diverse DeepInsight CNNs are the key winning factors of our team’s final submissions.

Thank you for reading through this long post, we hope that you gain something interesting and useful from it!