Supporting Information: 3D Convolutional Neural Networks and a CrossDocked Data Set for Structure-Based Drug Design

Paul G. Francoeur, Tomohide Masuda, Jocelyn Sunseri, Andrew Jia, Richard B. Iovanisci, Ian Snyder, and David R. Koes*

Department of Computational and Systems Biology, University of Pittsburgh, Pittsburgh, PA 15260

E-mail: dkoes@pitt.edu

Hyperparameter Search

More than 50 hyperparameters (Figure S1) for training and the network topology were explored both automatically and rationally using a variety of techniques. More details and code can be found at `https://github.com/gnina/scripts/tree/master/affinity_search`. Due to the inherent variability of model training, every parameter set was used to train five models with different random seeds on the PDBbind Refined set with the goal of improving the average clustered cross-validation affinity prediction and pose selection performance.

Def2017 was used as a baseline model and each parameter was individually varied relative to this model. Bayesian optimization[1] as implemented by the Spearmint package (`https://`

```
--base_lr_exp lr      Initial learning rate exponent, for log10 scaling
--momentum m          Momentum parameters, default 0.9
--weight_decay_exp w  Weight decay exponent (for log10 scaling)
--solver {SGD,Adam}   Solver type
--balanced {0,1}      Balance training data
--stratify_receptor {0,1} Stratify receptor
--stratify_affinity {0,1} Stratify affinity, min=2,max=10
--stratify_affinity_step {1,2,4} Stratify affinity step
--resolution {0.5,1.0} Grid resolution
--jitter j            Amount of jitter to apply
--loss_gap g          Affinity loss gap
--loss_penalty p      Affinity loss penalty
--loss_pseudohuber {0,1} Use pseudohuber loss
--loss_delta d        Affinity loss delta
--ranklossmult r      Affinity rank loss multiplier
--ranklossneg {0,1}   Affinity rank loss include neg
--pool1_size {0,2,4,8} Pooling size for layer 1
--pool1_type {MAX,AVE} Pooling type for layer 1
--conv1_func {ReLU,leaky,ELU,Sigmoid,TanH} Activation function in layer 1
--conv1_norm {BatchNorm,LRN,none}  Normalization for layer 1
--conv1_size {1,3,5,7} Convolutional kernel size for layer 1
--conv1_stride {1,2,3,4} Convolutional stride for layer 1
--conv1_width {0,1,2,4,8,16,32,64,128,256,512,1024} Convolutional output width for layer 1
--conv1_init {gaussian,positive_unitball,uniform,xavier,msra,radial,radial.5} Weight initialization for layer 1
--pool2_size {0,2,4,8} Pooling size for layer 2
--pool2_type {MAX,AVE} Pooling type for layer 2
--conv2_func {ReLU,leaky,ELU,Sigmoid,TanH} Activation function in layer 2
--conv2_norm {BatchNorm,LRN,none} Normalization for layer 2
--conv2_size {1,3,5,7} Convolutional kernel size for layer 2
--conv2_stride {1,2,3,4} Convolutional stride for layer 2
--conv2_width {0,1,2,4,8,16,32,64,128,256,512,1024} Convolutional output width for layer 2
--conv2_init {gaussian,positive_unitball,uniform,xavier,msra,radial,radial.5} Weight initialization for layer 2
--pool3_size {0,2,4,8} Pooling size for layer 3
--pool3_type {MAX,AVE} Pooling type for layer 3
--conv3_func {ReLU,leaky,ELU,Sigmoid,TanH} Activation function in layer 3
--conv3_norm {BatchNorm,LRN,none}  Normalization for layer 3
--conv3_size {1,3,5,7} Convolutional kernel size for layer 3
--conv3_stride {1,2,3,4}  Convolutional stride for layer 3
--conv3_width {0,1,2,4,8,16,32,64,128,256,512,1024}  Convolutional output width for layer 3
--conv3_init {gaussian,positive_unitball,uniform,xavier,msra,radial,radial.5}  Weight initialization for layer 3
--pool4_size {0,2,4,8}  Pooling size for layer 4
--pool4_type {MAX,AVE} Pooling type for layer 4
--conv4_func {ReLU,leaky,ELU,Sigmoid,TanH} Activation function in layer 4
--conv4_norm {BatchNorm,LRN,none}  Normalization for layer 4
--conv4_size {1,3,5,7}  Convolutional kernel size for layer 4
--conv4_stride {1,2,3,4}  Convolutional stride for layer 4
--conv4_width {0,1,2,4,8,16,32,64,128,256,512,1024}  Convolutional output width for layer 4
--conv4_init {gaussian,positive_unitball,uniform,xavier,msra,radial,radial.5}  Weight initialization for layer 4
--pool5_size {0,2,4,8}  Pooling size for layer 5
--pool5_type {MAX,AVE}  Pooling type for layer 5
--conv5_func {ReLU,leaky,ELU,Sigmoid,TanH}  Activation function in layer 5
--conv5_norm {BatchNorm,LRN,none}  Normalization for layer 5
--conv5_size {1,3,5,7}  Convolutional kernel size for layer 5
--conv5_stride {1,2,3,4}  Convolutional stride for layer 5
--conv5_width {0,1,2,4,8,16,32,64,128,256,512,1024}  Convolutional output width for layer 5
--conv5_init {gaussian,positive_unitball,uniform,xavier,msra,radial,radial.5}  Weight initialization for layer 5
--fc_affinity_hidden {0,16,32,64,128,256,512,1024,2048,4096}  Hidden nodes in affinity fully connected layer
--fc_affinity_func {ReLU,leaky,ELU,Sigmoid,TanH}  Activation function in for first affinity hidden layer
--fc_affinity_hidden2 {0,32,64,128,256,512,1024,2048,4096} Second set of hidden nodes in affinity fully connected layer
--fc_affinity_func2 {ReLU,leaky,ELU,Sigmoid,TanH}  Activation function in for second affinity hidden layer
--fc_affinity_init {gaussian,positive_unitball,uniform,xavier,msra} Weight initialization for affinity fc
--fc_pose_hidden {0,32,64,128,256,512,1024,2048,4096} Hidden nodes in pose fully connected layer
--fc_pose_func {ReLU,leaky,ELU,Sigmoid,TanH} Activation function in for first pose hidden layer
--fc_pose_hidden2 {0,32,64,128,256,512,1024,2048,4096} Second set of hidden nodes in pose fully connected layer
--fc_pose_func2 {ReLU,leaky,ELU,Sigmoid,TanH} Activation function in for second pose hidden layer
--fc_pose_init {gaussian,positive_unitball,uniform,xavier,msra} Weight initialization for pose fc
```

Figure S1: Hyperparameters explored when constructing and training models.

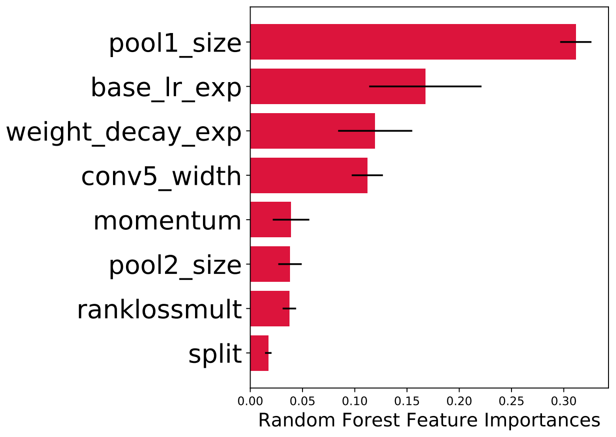


Figure S2: Feature importances of final random forest hyperparameter model.

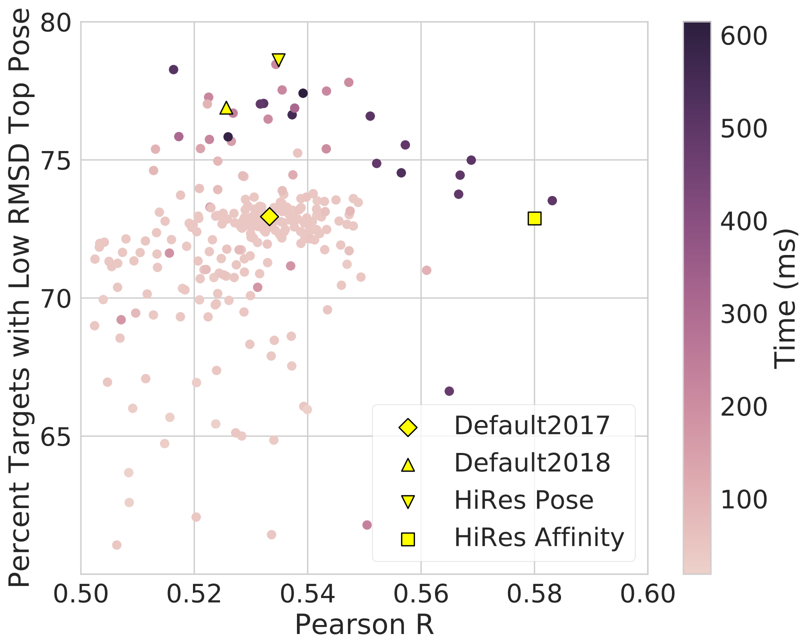


Figure S3: Landscape of all evaluated models. Every data point is the average clustered cross-validation performance of five models trained with the same hyperparameters and a different random seed.

`github.com/JasperSnoek/spearmint`) was used to construction a Gaussian process model of the hyperparameter space and suggest the most informative parameters to evaluate. We found this approach had difficulty scaling to large parameter spaces (solving the Gaussian process model took longer than training a neural network) so we applied this approach on subsets of the full parameter space.

We found a much more scalable approach was to train a random forest model to predict the performance of a model given the hyperparameters and then use a genetic algorithm to find hyperparameter sets that maximize performance according to this model. The feature importances of our final random forest model are shown in Figure S2. Relatively few features are more important than which split of the data is used.

The final landscape of all 382 evaluated models is shown in Figure S3. The main observation is that higher performing models tend to be more computationally intensive, which is undesirable for high-throughput applications.

Table S1: Training Hyper Parameters

Selected Training Hyper Parameters			
Data Set	step_when	step_end_cnt	percent_reduced
PDBbind Refined (Crystal)	5	4	100
PDBbind Refined (Core)	25	4	100
PDBbind Refined (CCV)	18	4	100
PDBbind General (Crystal)	5	4	100
PDBbind General (Core)	88	4	100
PDBbind General (CCV)	88	4	100
ReDocked (CCV)	200	3	3.82
CrossDocked (CCV)	200	3	0.132

PDB Identifiers

When comparing to previous results using the PDBbind Core set as a test set, we note there is a discrepancy between the Core sets reported by Jiménez Luna et al.[2] ($n = 290$), and what is currently reported as in the Core set by the PDBbind ($n = 285$). For reference, the PDB identifiers we used for all our datasets are available at `https://github.com/gnina/models` along with the raw data used to train models.

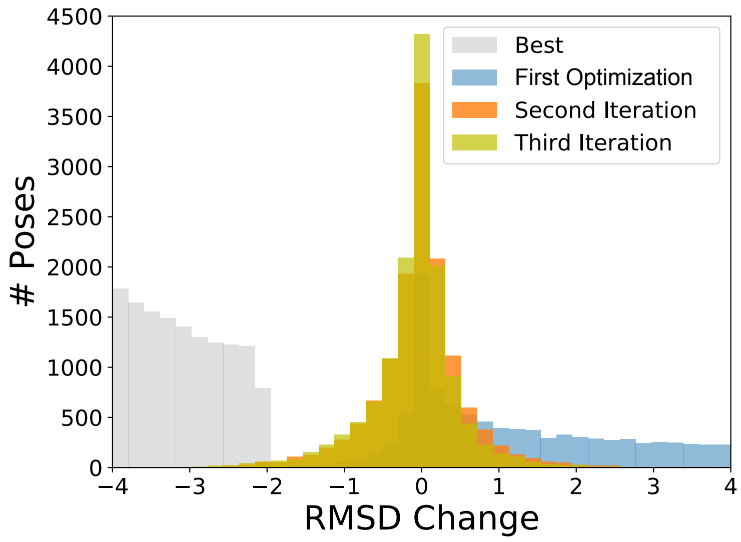


Figure S4: An example of the change in distribution of the change in pose RMSD relative to crystal pose after optimization by a CNN model. Only low RMSD (< 4Å poses are optimized. A perfect model would optimize the pose to equal the crystal pose resulting in the grey distribution.

Iterative Training

In order to generate counterexamples, we optimize training set poses with respect to a model trained on that training set. Optimization is performed by treating neural network gradients, backpropagated onto atom centers (as implemented by `libmolgrid`[3]), as forces and applying BFGS optimization, as implemented in gnina. As shown in Figure S4, the initial optimization results in poses that, while scored highly by the model, deviate significantly from the ideal pose. In fact, in the majority of cases, the RMSD of a pose is substantially worsened after optimization. Including these counterexamples in further rounds of training shifts the distribution so that the majority of optimized poses improve their RMSD.

We also tested the effect of the iterative poses on model performance on a docking run where the model was used in the energy minimization of the pose generation process. In Fig-

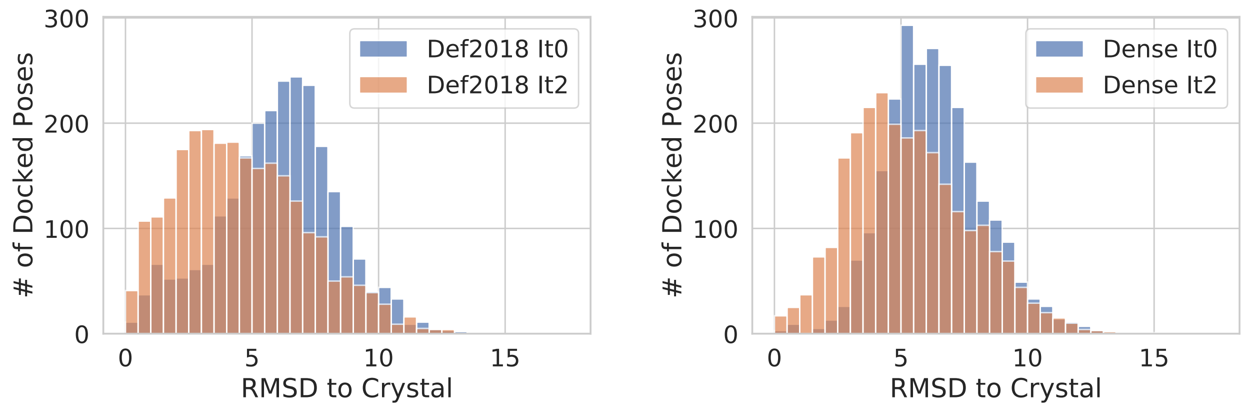


Figure S5: A Histogram of the RMSD of poses minimized using the Def2018 or DenseNet models trained with or without our counterexamples. While not as impressive as the Def2018 model from which the counterexamples are generated, they still yield a positive benefit for the DenseNet.

ure S5 we show the performance of redocking the PDBbind core set when using the Def2018 or Dense models, each trained with or without the iteratively generated counterexamples. Note we show the distribution of all docked poses, not just the top scoring pose. The inclusion of counterexamples during training results in more low RMSD poses. The distribution shift is more significant for the Def2018 models (mean RMSD 6.03 to 4.62) than with the Dense models (mean RMSD 6.35 to 5.33). This is expected as the counterexamples are specific to the Def2018 model. However, we still observe a benefit with the Dense model.

Extended Ligand-Only Analysis

During our training procedure we label our poses as good (under 2Å RMSD from the crystal pose) or bad (over 2Å RMSD from the crystal pose), in order to have our models learn to identify binding modes. Chemically speaking, the binding mode of a given protein-ligand complex is determined by the interactions between the receptor and the ligand. When we train a ligand-only model, these types of interactions are lost. Therefore, if there is no other information that contributes to what identifies a pose as "good" or "bad", we expect the distribution of CNNscores to be random centered around 0.5. However, a model trained with the complete complex would be able to identify these interactions. Ergo, we would expect such a model to have a bi-modal distribution that separates the "good" from "bad" poses. As shown in Figure S6, the performance of the Def2018 model trained without a receptor is close to what we expected. However, there is some signal present in the ligand information that allows for a skewed distribution of CNNscores. It should be noted that the CrossDocked set when trained with the receptor is not a balanced bi-modal distribution due to each pocket-ligand complex being dominated by negative examples.

In order to investigate this effect further, we analyzed the fraction of poses of each receptor that are under 2Å RMSD from the crystal pose as a function of the CNNscore for a given complex in Figure S7. For models trained without the receptor, as the fraction of good poses for the complex rises, the CNNscores also rise. This provides additional support that there is some signal in only the information present in the ligand to allow for some discrimination between good and bad poses.

Additionally, we compared the performance of these CNN models trained with and without receptor information to the performance of an assortment of other common regression models that were trained using the 1D simple chemical descriptors utilized for construction of the DUD-E dataset.[4] The descriptors are molecular weight, number of hydrogen bond acceptors, number of hydrogen bond donors, number of rotatable bonds, logP, and net charge. The models assessed were linear regression with Lasso regularization, K-nearest neighbors, a decision tree, a random forest, gradient boosted decision trees, and support vector regression; a comparison between these models and the CNN models trained with and without receptor information is shown in Table S3. Basic grid search hyperparameter optimization was performed for these methods using the PDBbind Refined set. Notably, the approach based on training standard models using the DUD-E chemical descriptors yields performance that compares favorably with the CNN models when considering the PDBbind General set, but performance on the CrossDocked set is significantly worse when information is limited to these descriptors. This further suggests that the CrossDocked set is more challenging for machine learning approaches than existing benchmarks based solely on the PDBbind datasets.

References

(1) Snoek, J.; Larochelle, H.; Adams, R. P. Practical bayesian optimization of machine learning algorithms. Advances in neural information processing systems. 2012; pp 2951–2959.

(2) Jiménez Luna, J.; Skalic, M.; Martinez-Rosell, G.; De Fabritiis, G. K DEEP: Protein-ligand absolute binding affinity prediction via 3D-convolutional neural networks. *J. Chem. Inf. Model.* **2018**,

(3) Sunseri, J.; Koes, D. R. libmolgrid: Graphics Processing Unit Accelerated Molecular Gridding for Deep Learning Applications. *J. Chem. Inf. Model.* **2020**, *60*, 1079–1084, PMID: 32049525.

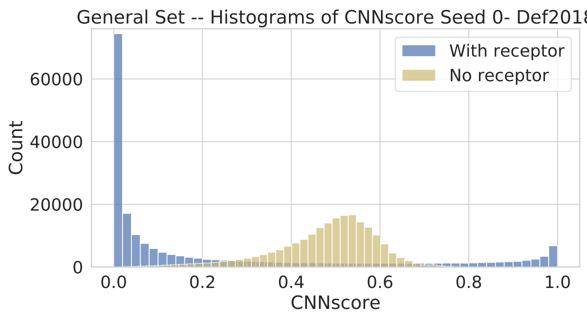
(4) Mysinger, M. M.; Carchia, M.; Irwin, J. J.; Shoichet, B. K. Directory of Useful Decoys, Enhanced (DUD-E): Better Ligands and Decoys for Better Benchmarking. *J. Med. Chem.* **2012**, *55*, 6582–6594.

Table S2: Comparison of ReducedCrossDocked2020 to CrossDocked2020.

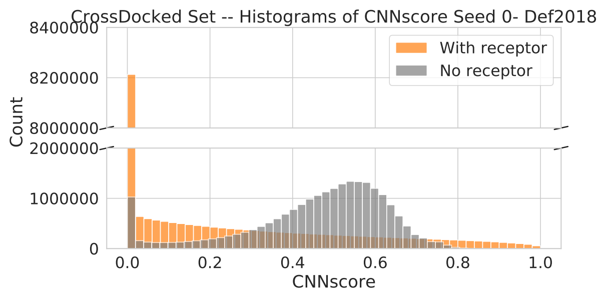
Data Set	Model	RMSE	R	AUC	Top1	BP	Rand
CrossDocked2020	Def2018	1.47	0.58	0.91	0.54	0.97	0.03
ReducedCrossDocked2020	Def2018	1.45	0.59	0.89	0.82	1.0	0.30

Table S3: Comparison of CNN models trained with and without receptor information and a variety of models trained with simple chemical descriptors. R and RMSE values are the mean across the ensemble.

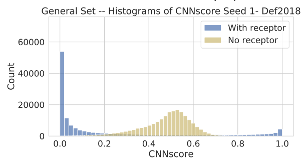
Dataset	Model	RMSE	R
General (CCV)	CNN (With Receptor)	1.65	0.56
General (CCV)	Gradient Boosted Trees (Descriptors)	1.63	0.54
General (CCV)	Random Forest (Descriptors)	1.65	0.52
General (CCV)	CNN (Without Receptor)	1.72	0.52
General (CCV)	Decision Tree (Descriptors)	1.69	0.50
General (CCV)	KNN (Descriptors)	1.70	0.50
General (CCV)	SVM (Descriptors)	1.69	0.49
General (CCV)	Lasso (Descriptors)	1.70	0.48
CrossDocked (CCV)	CNN (With Receptor)	1.47	0.58
CrossDocked (CCV)	CNN (Without Receptor)	1.58	0.49
CrossDocked (CCV)	Gradient Boosted Trees (Descriptors)	1.82	0.31
CrossDocked (CCV)	Random Forest (Descriptors)	1.82	0.30
CrossDocked (CCV)	SVM (Descriptors)	1.92	0.28
CrossDocked (CCV)	KNN (Descriptors)	1.86	0.25
CrossDocked (CCV)	Lasso (Descriptors)	1.86	0.24
CrossDocked (CCV)	Decision Tree (Descriptors)	1.93	0.23



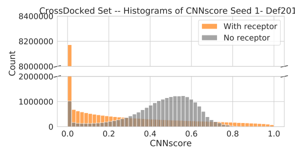
(a) Seed 0 General



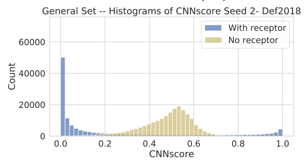
(b) Seed 0 CrossDocked



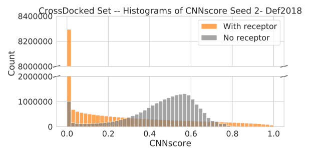
(c) Seed 1 General



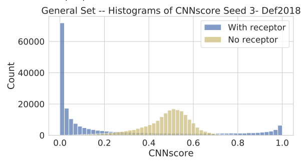
(d) Seed 1 CrossDocked



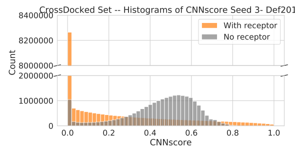
(e) Seed 2 General



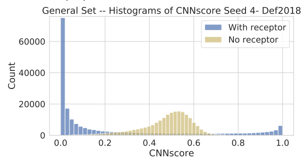
(f) Seed 2 CrossDocked



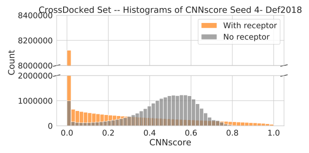
(g) Seed 3 General



(h) Seed 3 CrossDocked

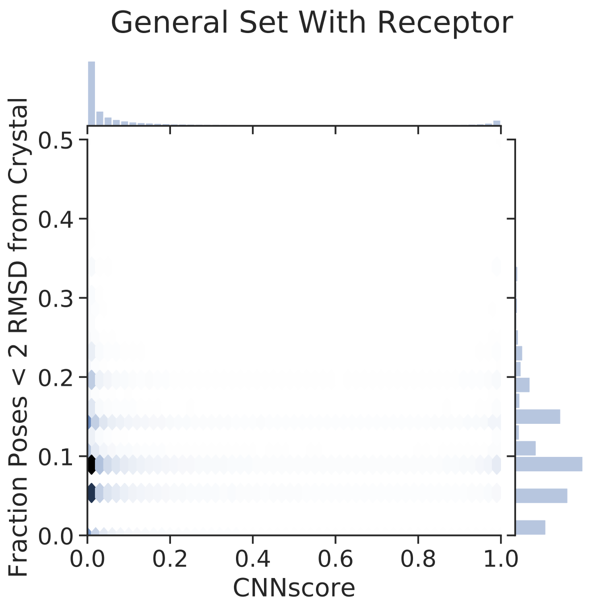


(i) Seed 4 General

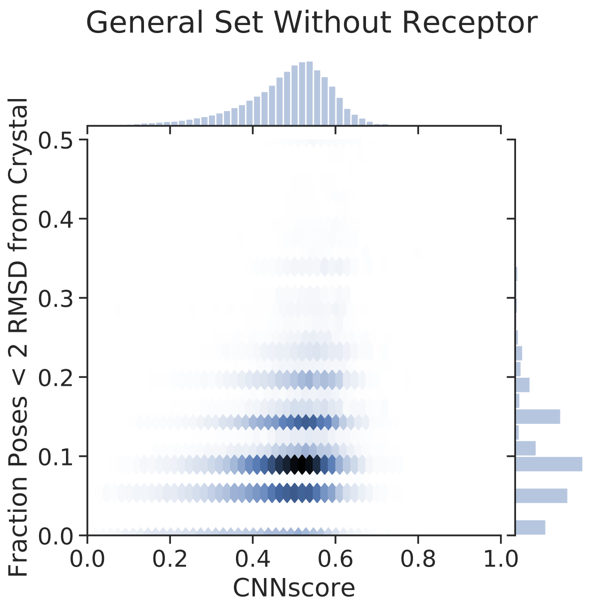


(j) Seed 4 CrossDocked

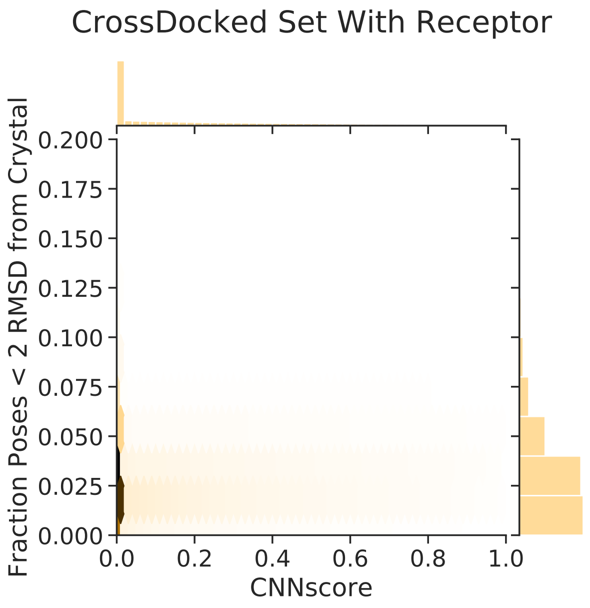
Figure S6: CNNscore distributions of ligand-only and regular models trained with CCV General and CrossDocked sets. Distributions for all five trained models, which were trained with different random seeds, are shown.



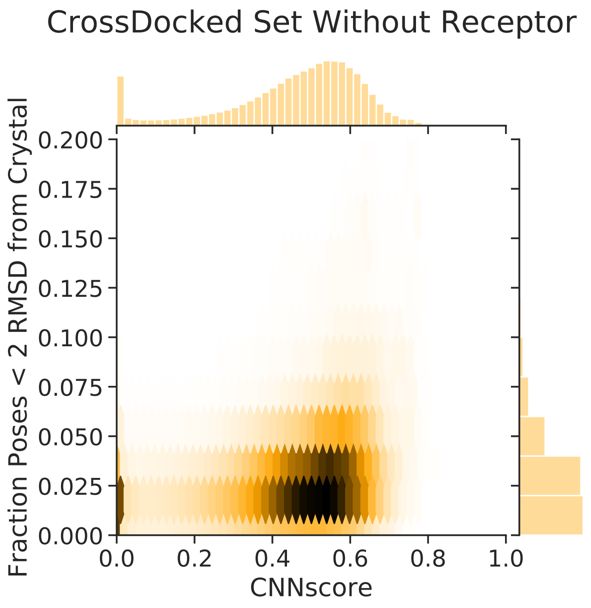
(a) CCV General set - trained with receptor information.



(b) CCV General set - trained without receptor information.



(c) CCV CrossDocked Set - trained with receptor information.



(d) CCV CrossDocked Set - trained without receptor information.

Figure S7: Plots of the fraction of good poses for a given complex as a function of CNNscore for the Def2018 model. The results of 1 model (seed 0) is shown per plot.

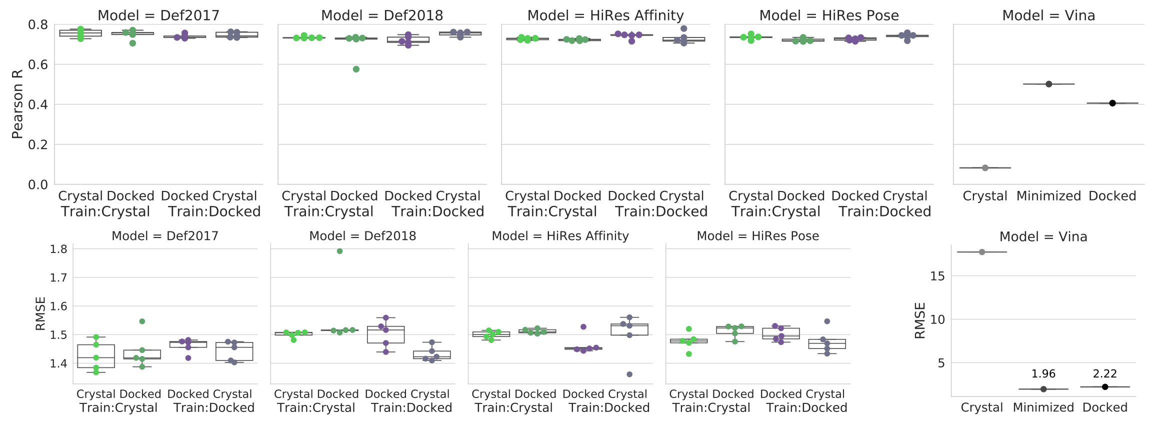
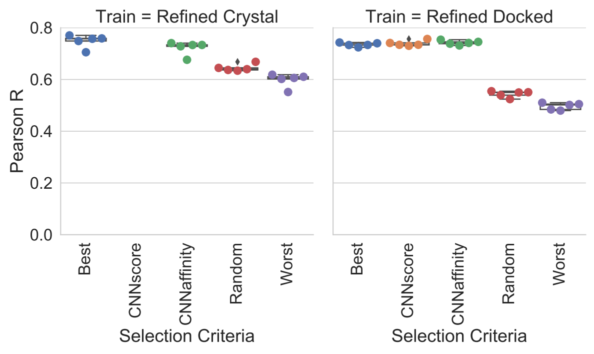
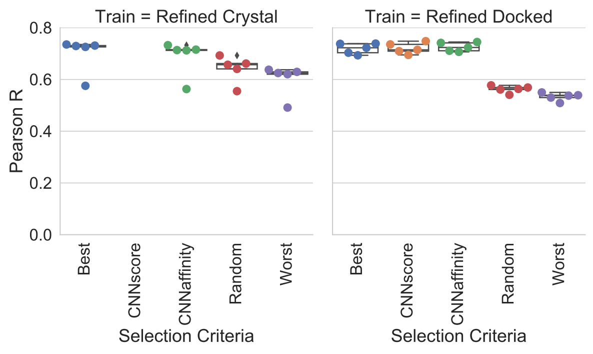


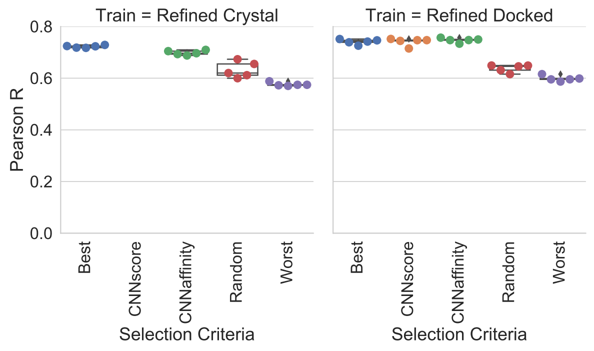
Figure S8: Affinity prediction correlation and RMSE on PDBbind Core set for models trained using crystal or docked poses from the Refined Set. Autodock Vina was used as a baseline. The test set consisted of either crystal or docked poses. Note: there is an increased scale for the Autodock Vina RMSE results plot



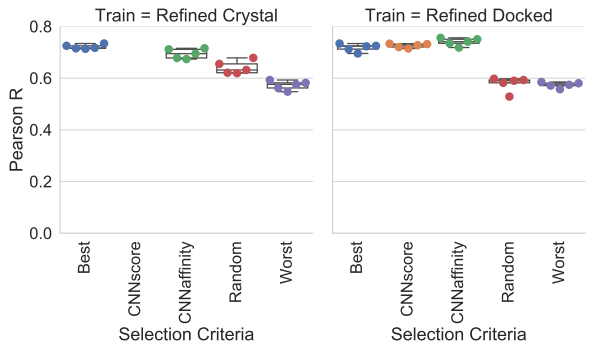
(a) Def2017 - Affinity Prediction



(b) Def2018 - Affinity Prediction

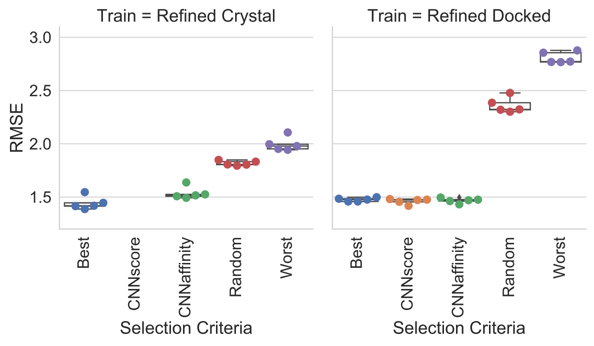


(c) HiRes Affinity - Affinity Prediction

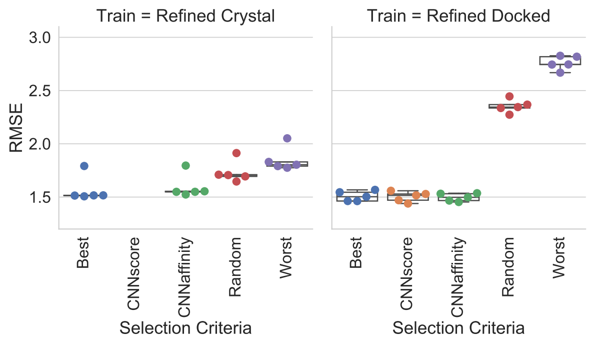


(d) HiRes Pose - Affinity Prediction

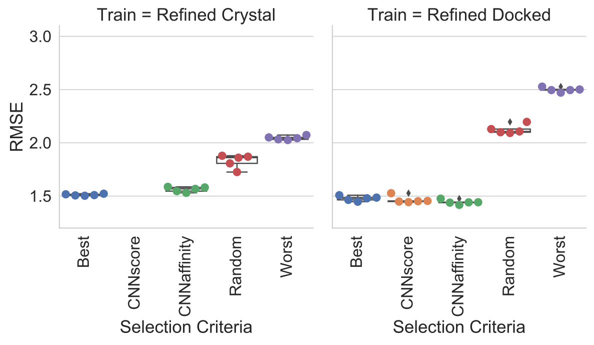
Figure S9: Pearson correlation of affinity prediction performance across all models with different pose selection methods when trained on Crystal or Docked poses of PDB Refined and tested on Core. 'Best' is the lowest RMSD pose to the crystal pose, CNNscore is the highest predicted scoring pose (not applicable for Crystal trained models), CNNaffinity is the highest predicted affinity, 'Worst' is the highest RMSD pose to the crystal pose, and Random is taking a pose at random.



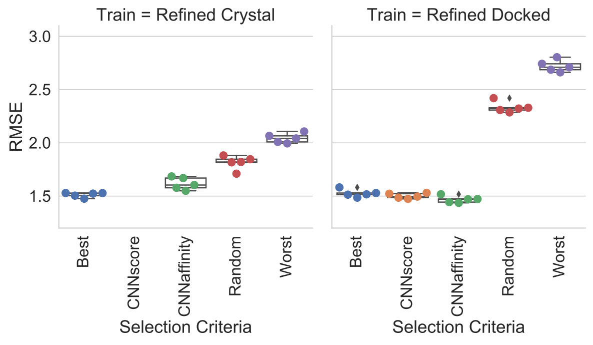
(a) Def2017 - Predicted Affinity Error



(b) Def2018 - Predicted Affinity Error

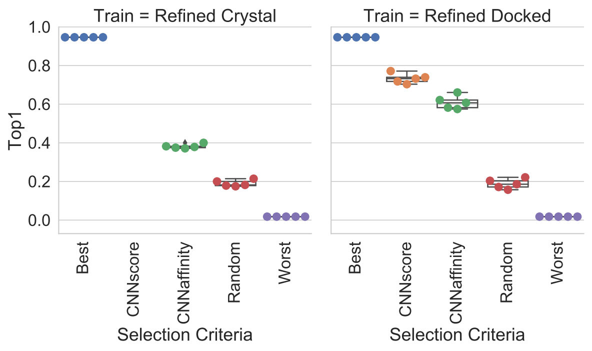


(c) HiRes Affinity - Predicted Affinity Error

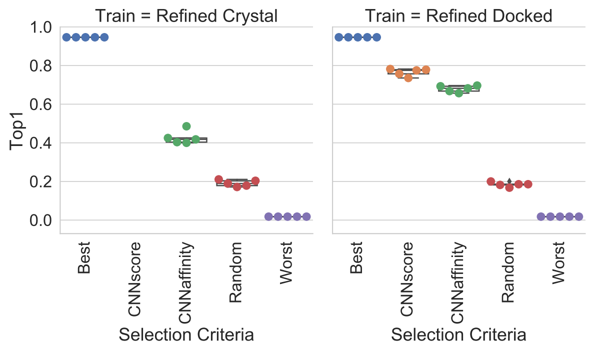


(d) HiRes Pose - Predicted Affinity Error

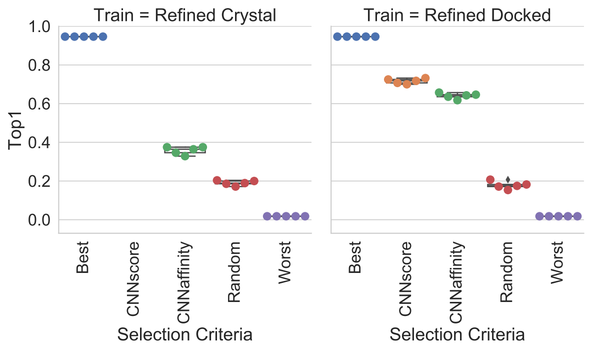
Figure S10: RMSE error of affinity prediction performance across all models with different pose selection methods when trained on Crystal or Docked poses of PDB Refined and tested on Core. 'Best' is the lowest RMSD pose to the crystal pose, CNNscore is the highest predicted scoring pose (not applicable for Crystal trained models), CNNaffinity is the highest predicted affinity, 'Worst' is the highest RMSD pose to the crystal pose, and 'Random' is taking a pose at random.



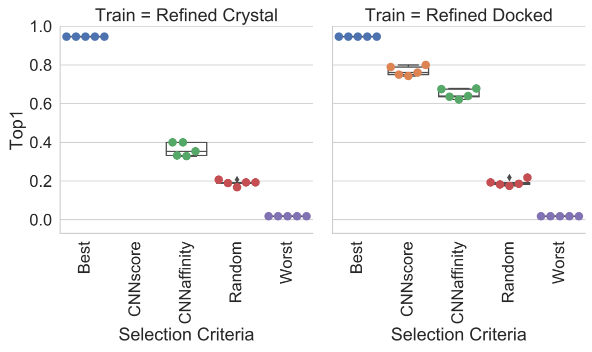
(a) Def2017 - Pose Selection



(b) Def2018 - Pose Selection

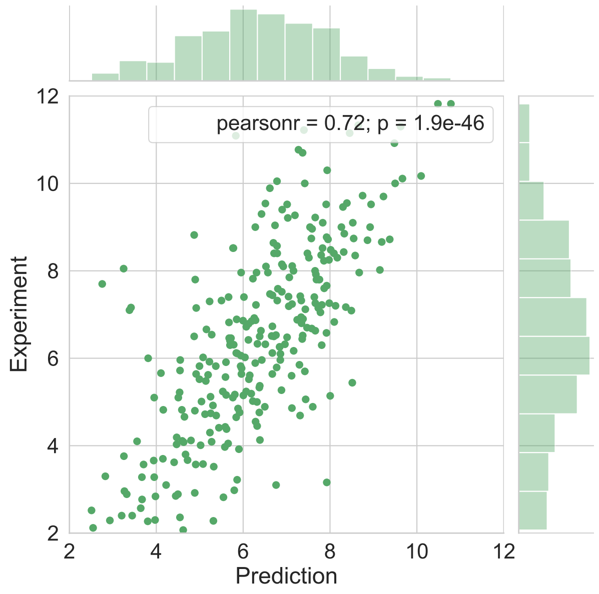


(c) HiRes Affinity - Pose Selection

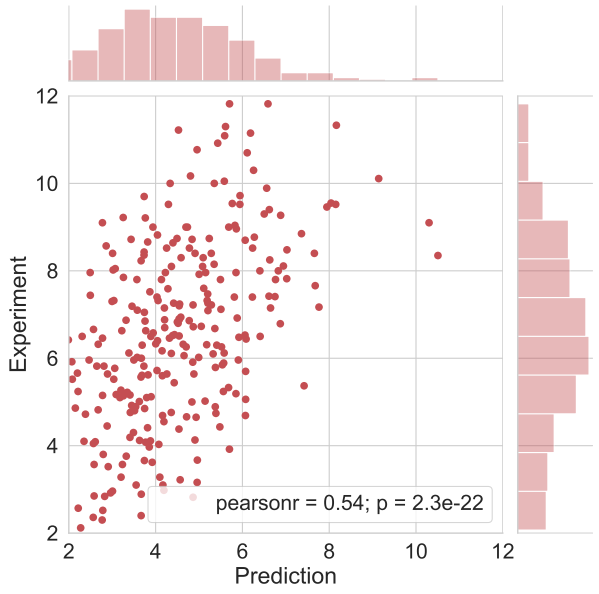


(d) HiRes Pose - Pose Selection

Figure S11: Performance of various pose selection methods when trained on Crystal or Docked poses of PDB Refined and tested on Core.

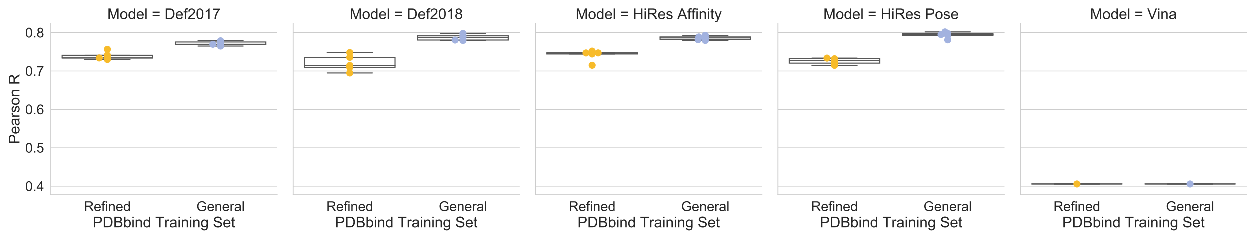


(a) Affinity prediction correlation using best available docked pose

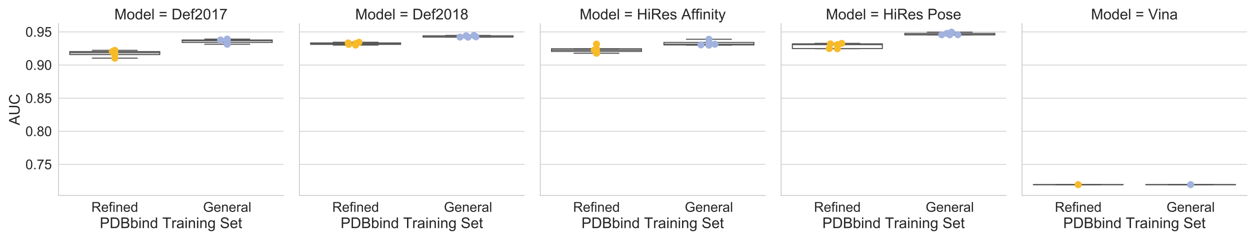


(b) Affinity prediction correlation using worse available docked pose

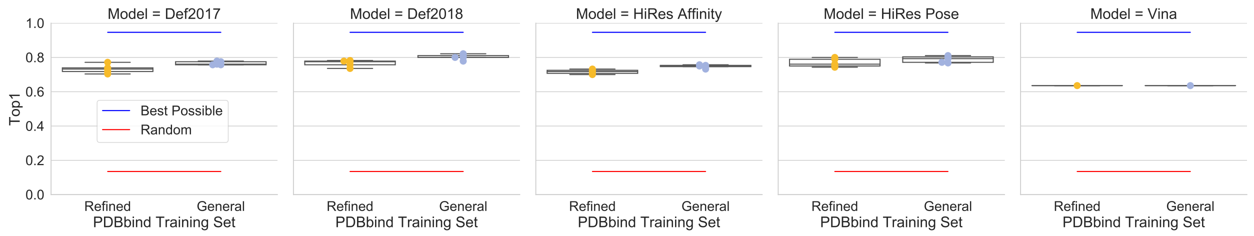
Figure S12: Correlation of affinity predictions of the Core set using the Def2018 model trained on PDBbind Refined when using the (a) best and (b) worst available docked pose of a ligand to make the prediction.



(a) Affinity prediction correlation

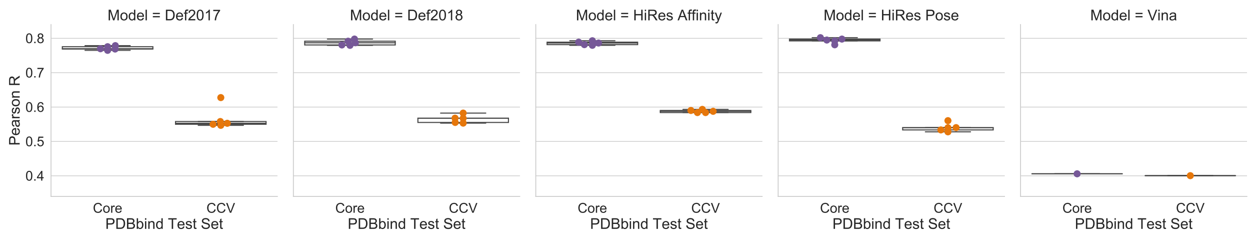


(b) Inter-target pose ranking

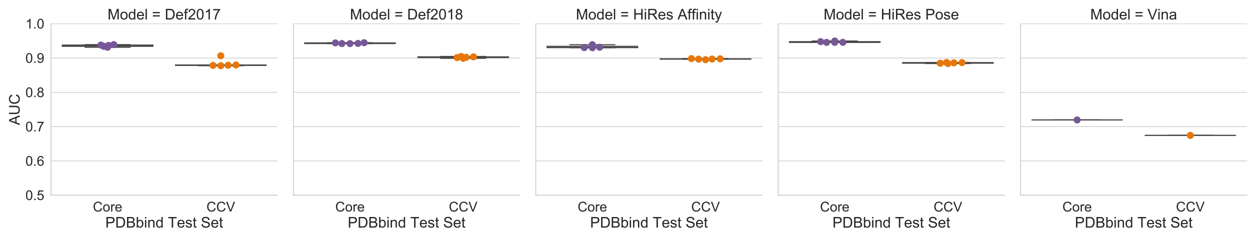


(c) Intra-target pose ranking

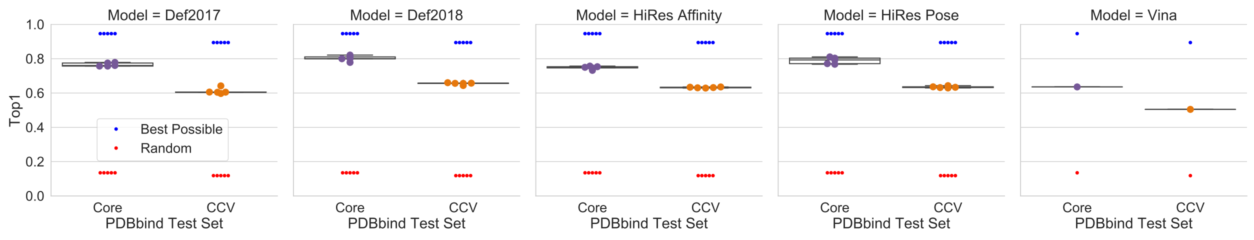
Figure S13: Performance on Core when the training set is expanded from PDB Refined to General.



(a) Affinity prediction correlation

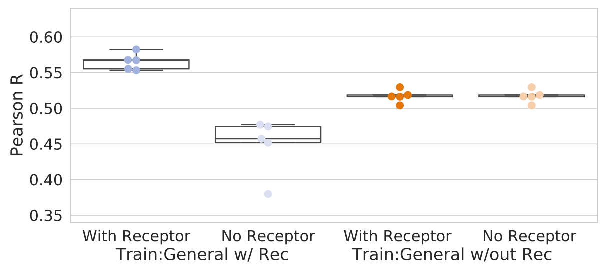


(b) Inter-target pose ranking

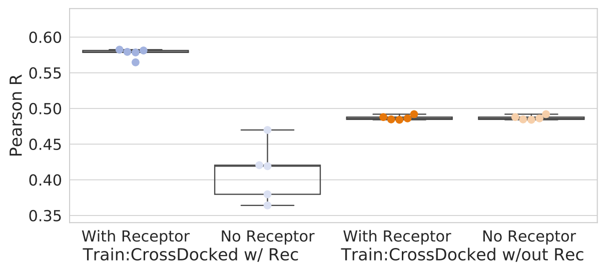


(c) Intra-target pose ranking

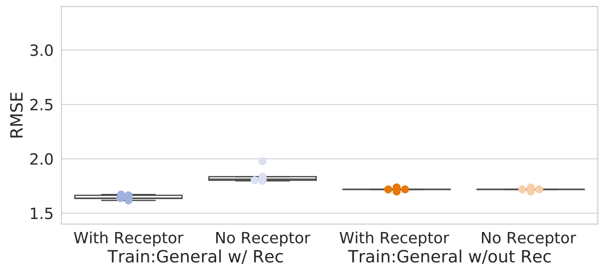
Figure S14: Performance when utilizing different train/test splits. Models were either trained on PDBbind General and tested on PDBbind Core (Core) or trained with clustered cross-validation splits of the PDBbind General. Note the same data is in both sets, but is divided differently among train and test.



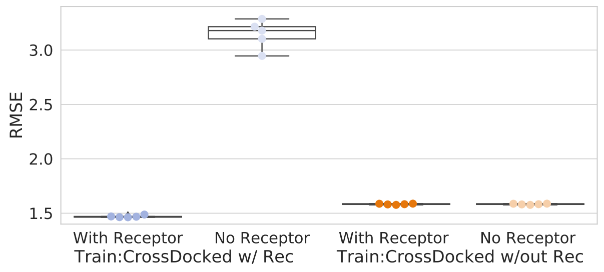
(a) PDBbind General affinity prediction correlation



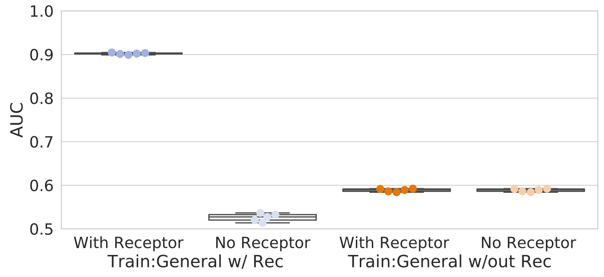
(b) CrossDock affinity prediction correlation



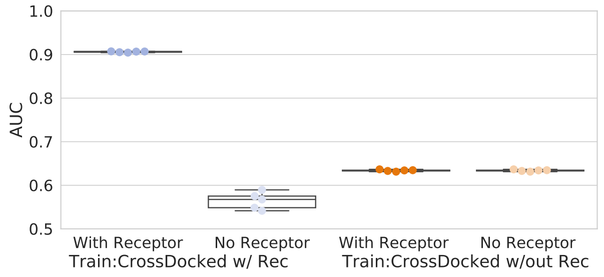
(c) PDBbind General affinity prediction RMSE



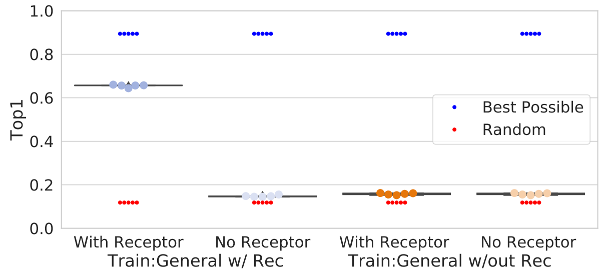
(d) CrossDock affinity prediction RMSE



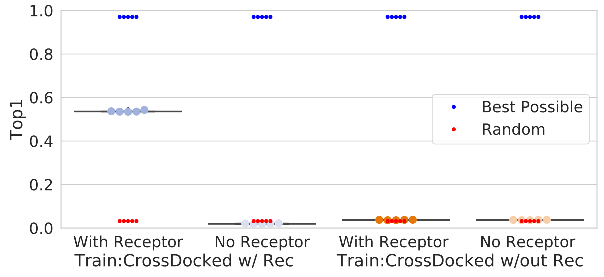
(e) PDBbind General inter-target pose ranking



(f) CrossDock inter-target pose ranking



(g) PDBbind General intra-target pose ranking



(h) CrossDock intra-target pose ranking

Figure S15: Ligand-only model performance. Def2018 models were trained with or without receptors (w/ Rec or w/out Rec) and evaluated on test sets with or without receptors (With Receptor or No Receptor).

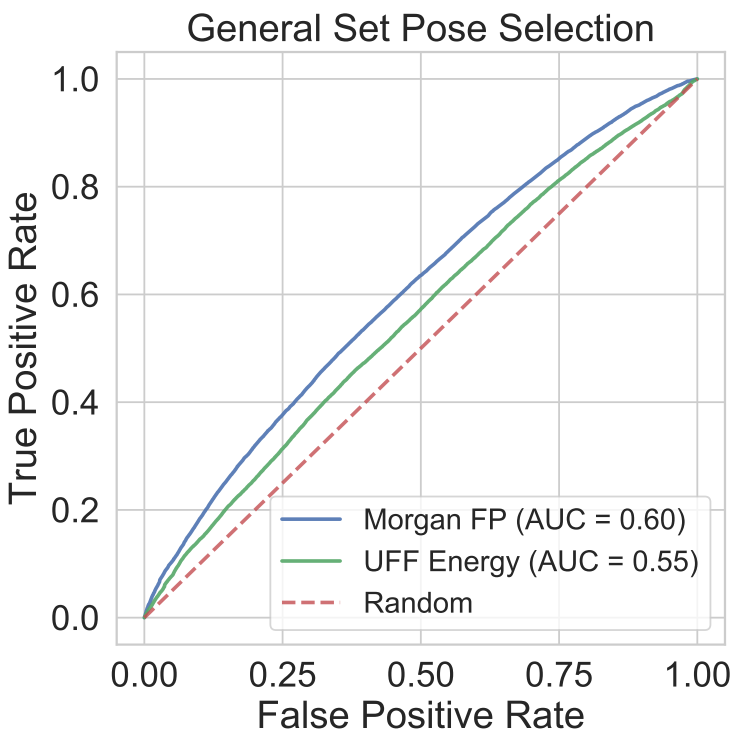
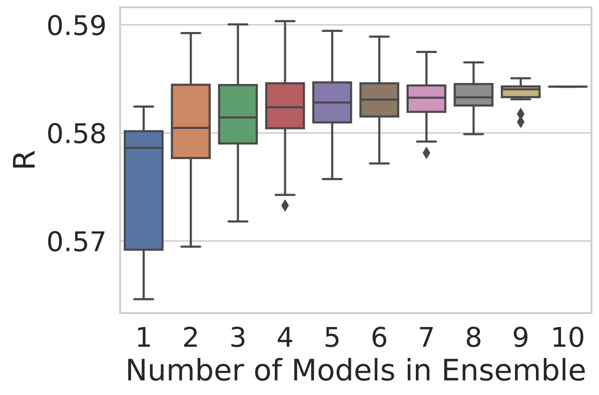
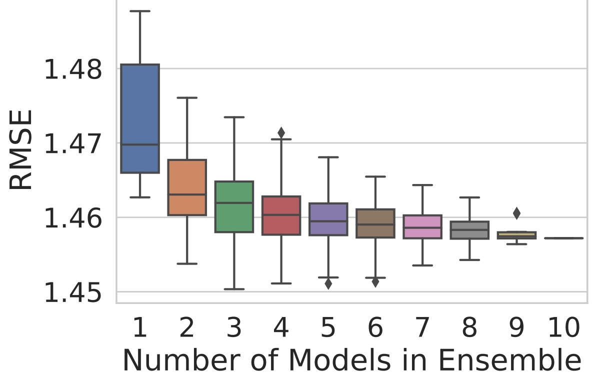


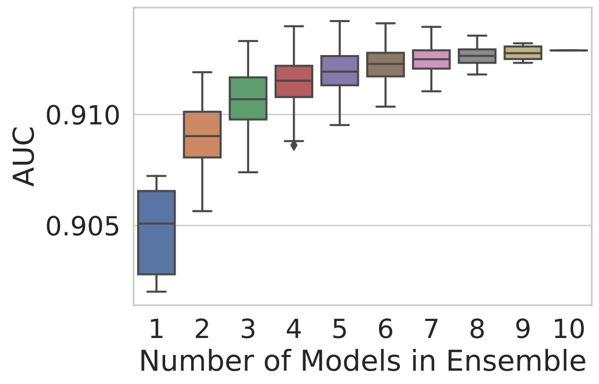
Figure S16: Investigating performance of using simple ligand-only classifiers to distinguish good from bad poses for the PDBbind General and Core sets.



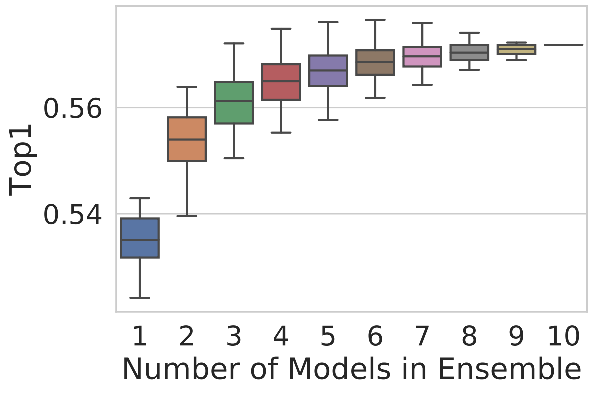
(a) CrossDocked2020 Pearson R from various model ensembles



(b) CrossDocked2020 RMSE from various model ensembles

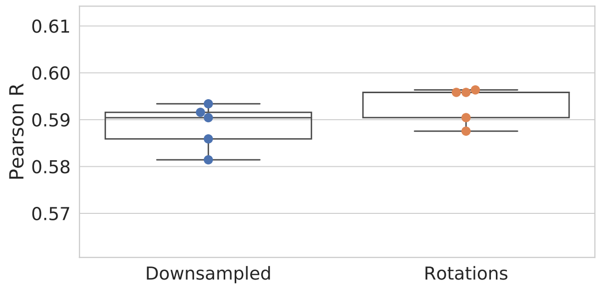


(c) CrossDocked2020 AUC from various model ensembles

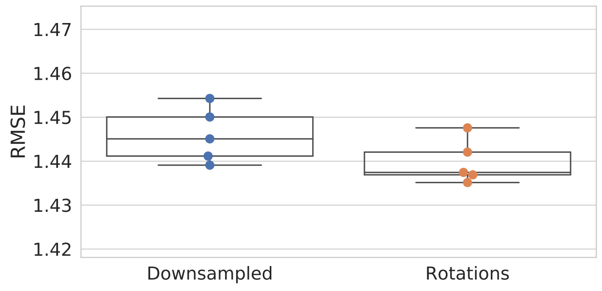


(d) CrossDocked2020 Top1 from various model ensembles

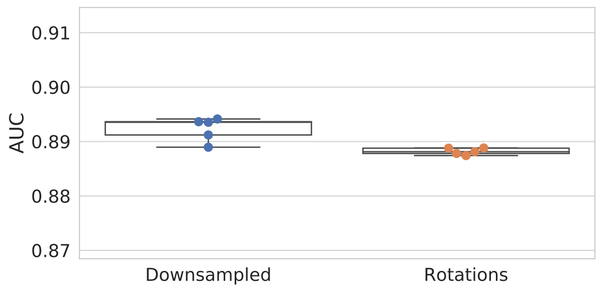
Figure S17: Ensembles of Default 2018 models trained on CrossDocked2020. There are diminishing returns after an ensemble of 5 models is used.



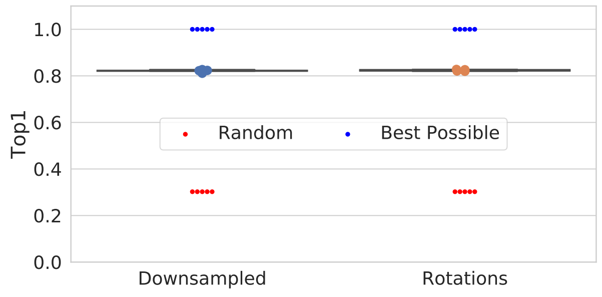
(a) ReducedCrossDocked2020 PearsonR from various model ensembles



(b) ReducedCrossDocked2020 RMSE for 10 rotation ensembles



(c) CrossDocked2020 AUC for 10 rotation ensembles



(d) CrossDocked2020 Top1 for 10 rotation ensembles

Figure S18: Using an Ensemble of Rotations for Default 2018 models trained on Reduced-CrossDocked2020.