1 Machine Learning Coursework Report

This is the report for COMP3055 Machine Learning Coursework. The coursework aims to practice various Machine Learning (ML) techniques on CIFAR10 dataset [1]. In this report, the detailed hardware settings and key python packages are showed in Section 2. Then, the implementation details and design choices for Task 1, Task 2 and Task 3 are illustrated in Section 3, Section 4 and Section 5 respectively. Finally, an empirical comparison to discuss the strength and weakness of Multi-Layer Perception (MLP) and Convolution Neural Network (CNN) approaches is in Section 6.

2 Hardware & Software Settings

Hardware Settings: AMD Ryzne 7 5800H CPU, NVIDIA Geforce RTX 3060 GPU.
Python Environment: python==3.7, cudatoolkit==11.3.1, cuDnn==8.0, pytorch==1.10.0, torchvision==0.11.1, sklearn ==0.0

3 Task 1

The Principal Component Analysis (PCA) is achieved by using the *sklearn.decompoisition* package. By changing the parameter *n_ components* in the PCA constructor, which specifies the number of components a PCA reduction need to keep in the original data, we can achieve different amount of information to keep. In my implementation, I apply PCA reduction on the original CIFAR10 dataset through batches (i.e., 64). My empirical study shows that, when the *n_ components* are set to the exponential of 2, the percentages of information kept in the PCA output feature vectors are close to linear (i.e., with an approximate gap of 12% in information kept for adjacent parameters). For each configuration, the PCA reduction is conducted 10 times on the original data and the averaged results are showed in Table 3.

Table 1: Information kept in feature vectors in different *n_ components* configurations

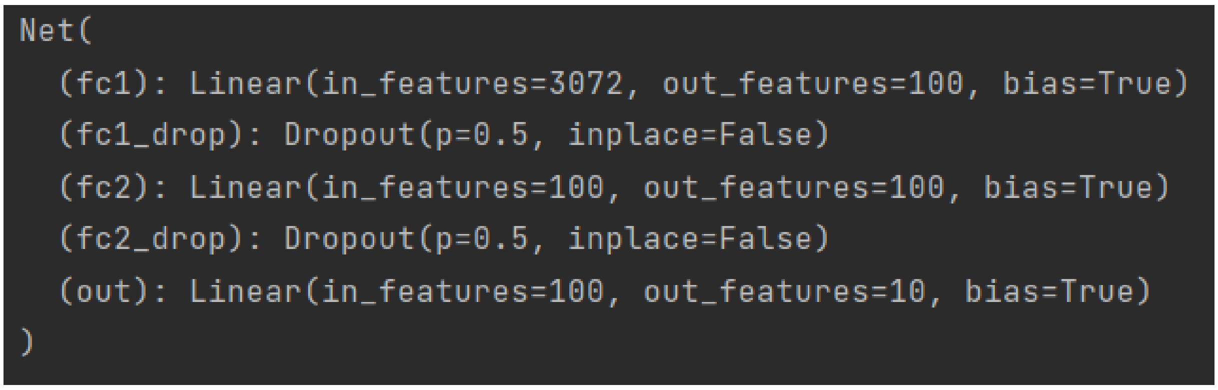
n_ components	1	2	4	8	16
Information Kept (%)	31.66	45.46	57.80	70.92	82.48

4 Task 2

In this section, I will showcase the implementation details of Task 2, which requires to design a MLP model for object detection on CIFAR10 dataset using 10-fold cross-validation in various hyperparameter settings. In this section, I will compare the effects of different learning rates (lr), levels of kept information and hidden node (hd) number on test accuracy and f1 values (i.e., both overall value and breakdown f1 value for each class).

4.1 Model Structure

For the MLP model, the structure and detailed parameters for each layer are shown in Figure 1. The model is simple as it contains one hidden layer, with 100 hidden nodes for



```
Net(
  (fc1): Linear(in_features=3072, out_features=100, bias=True)
  (fc1_drop): Dropout(p=0.5, inplace=False)
  (fc2): Linear(in_features=100, out_features=100, bias=True)
  (fc2_drop): Dropout(p=0.5, inplace=False)
  (out): Linear(in_features=100, out_features=10, bias=True)
)
```

Figure 1: The structure of my MLP model

Table 2: The averaged training, validation and test accuracy of my MLP model on original CIFAR10 dataset using 10-fold cross-validation, in different learning rates.

Learning Rates	0.001	0.005	0.01	0.015
Train Accuracy	52.47%	51.29%	47.91%	40.82%
Validation Accuracy	51.29%	48.85%	46.02%	41.17%
Test Accuracy	51.12%	48.99%	46.14%	40.93%

learning and inference. For the input layer and the hidden layer, I add two dropout layer with dropout ratio of 0.5 to prevent the model from overfitting.

4.2 Changing Learning Rates

First, I examine the effects of different learning rates (lr) on our model with varied configurations, which are 0.001, 0.005, 0.01, 0.015. The number of hidden nodes in the hidden layer is set to 100 and the input data is the original CIFAR10 data. The number of epoch for training is set to 100. Table 2 reports the experimental results in terms of the averaged training, validation and test accuracy among 10 sub-folds. Figure 2 reports the overall validation and test f1 scores for 10 sub-folds and Figure 3 reports the f1 scores on test set for each class.

From the results, we make 2 observations. First, the f1 score and test accuracy have negative correlation with lr (i.e., the higher the lr, the less the test accuracy). The MLP

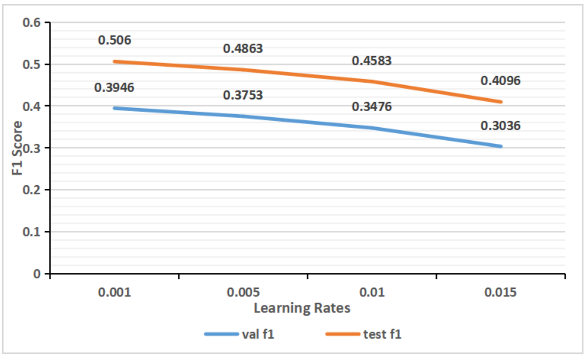


Figure 2: Overall f1 scores for validation and test set of my MLP model

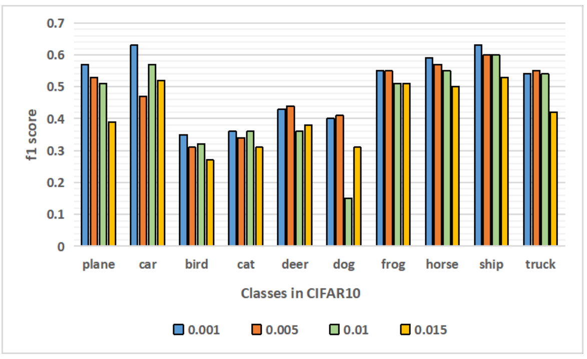


Figure 3: Micro f1 scores for test set of my MLP model

Table 3: The averaged training, validation and test accuracy of my MLP model on using 10-fold cross-validation with different levels of PCA feature reductions.

n_ components	1	2	4	8	16
Train Accuracy	15.81%	19.95%	25.23%	28.33%	30.90%
Val Accuracy	16.20%	20.70%	27.29%	31.73%	35.30%
Test Accuracy	16.26%	21.06%	28.20%	32.41%	35.92%

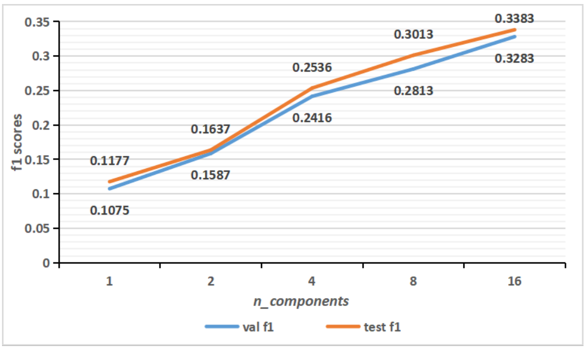


Figure 4: Overall f1 scores for validation and test set of my MLP model

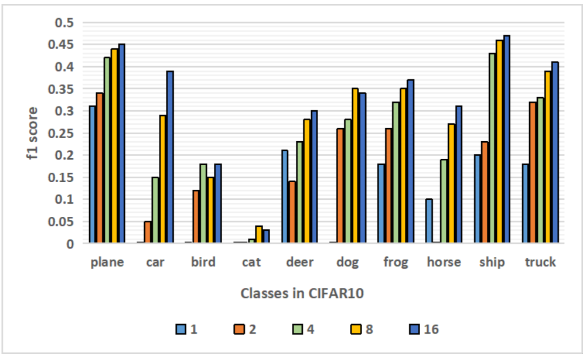


Figure 5: Micro f1 scores for test set of my MLP model

model has the best test accuracy when lr equals to 0.001. Second, the differences among training accuracy, validation accuracy and test accuracy are negligible, which indicates the model does not overfit. This can be attributed to the 10-cross validation which effectively prevents model overfitting on the training data.

4.3 Changing reduced input vector dimensions

Then, I aim to demystify the correlation between test accuracy of my MLP model with different PCA-reduced input feature vectors. In this experiment, the number of hidden nodes is set to 100, the learning rates is set to 0.001 which is the learning rates that train the best MLP model in the above section. The training epoch is set to 100. The configurations for *n_ components* is same as the configurations in the Task 1, since adjacent *n_ components* configurations results in linear information loss in percentages. Note that the number of nodes in the first layer of my MLP will be adaptively set to *n_ components*, when PCA is enabled. Table 3 reports the averaged training, validation and test accuracy of my MLP model on using 10-fold cross-validation with different levels of PCA feature reductions. Figure 4 reports the overall validation and test f1 scores and Figure 5 reports the f1 scores for each class for each configuration.

From the results, we make two observations. First, the f1 score and test accuracy have positive correlation with *n_ components*. This because higher value of *n_ components* keep more information in the original data, which provides sufficient features for model training and estimation, thus leading to the improvements of the performance. The second observations is similar to the second observation in the above section, which shows that 10-cross validation also prevents overfitting in this case.

Table 4: The averaged training, validation and test accuracy of my MLP model on using 10-fold cross-validation with different number of hidden nodes in the hidden layer

num of hidden nodes	50	100	150	200	300	400
Train Accuracy	39.99%	46.22%	49.43%	51.89%	54.92%	57.10%
Val Accruacy	44.96%	48.91%	50.98%	52.21%	53.68%	54.51%
Test Accuracy	44.90%	49.25%	51.49%	52.52%	53.79%	54.46%

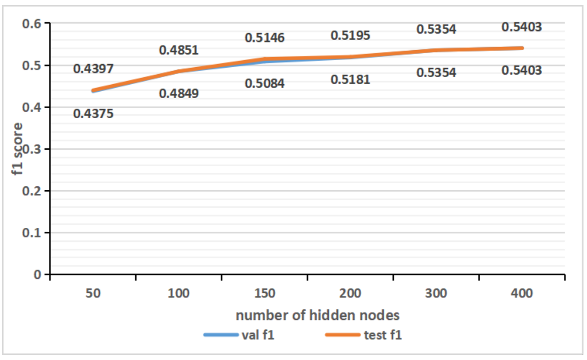


Figure 6: Overall f1 scores for validation and test set of my MLP model

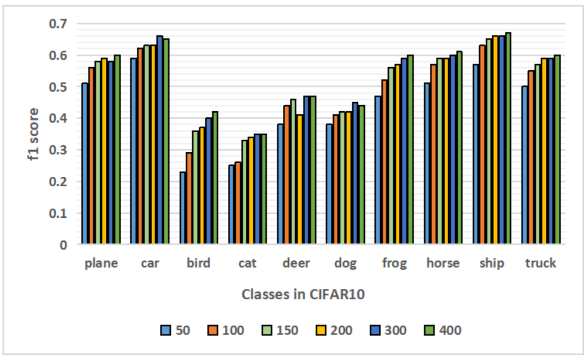


Figure 7: Micro f1 scores for test set of my MLP model

4.4 Changing the number of Hidden Nodes

Finally, in this section, I will present how the number of hidden nodes in the hidden layer affect the model accuracy. In this experiment, the learning rate is set to 0.001 and the input data is the original data. The training epoch is set to 100. Table 4 reports the averaged training, validation and test accuracy of my MLP model on using 10-fold cross-validation with different number of hidden nodes in the hidden layer. Figure 6 reports the overall f1 scores[1] and Figure 7 reports the f1 scores for each class with different number of hidden nodes.

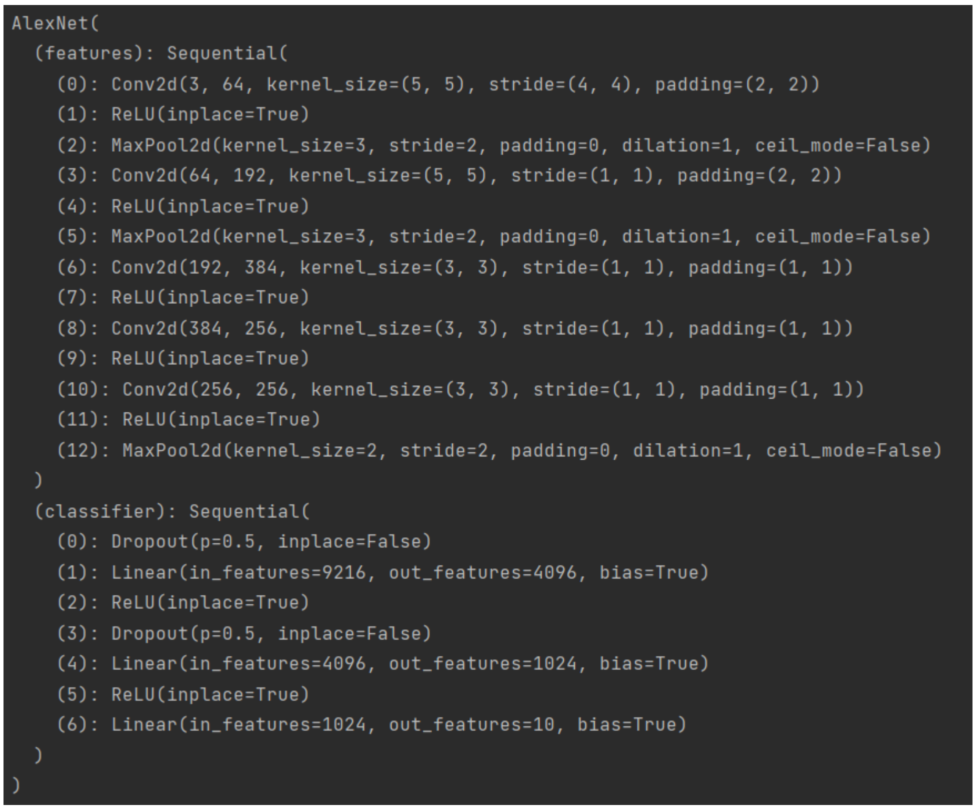
From the results, we make two observations. First, higher number of hidden nodes achieves better model performance. The MLP model achieves best test accuracy (i.e.,54.46%) when the number of hidden nodes is 400. Second, the model have slight overfitting when when the number of hidden nodes is 400.

5 Task 3

For Task 3, I use the AlexNet [2] as the main Convolutional Neural Network (CNN) model for object detection, which contains five convolutional layers for learning and inference. The detailed model structure is showed in Figure 8. The input data is first cropped into size of 224*224*3, which is the dimension of the input data in the original AlextNet. Cropping the input data from size of 32*32*3 to 224*224*3 is a way of data augmentation, which can improve the model learning capability and prevent over-fitting. My model is different from the original AlexNet model as the kernel size of the first convolutional layer for my model is 5 while the kernel size of first conv-layer of the original AlexNet is 11. My empirical study shows that by changing the size of kernel size in the first layer can obtain an average of 3% test accuracy improvements. In my implementation, the training

[1]The curves for val f1 and test f1 are overlapped due to close values

epoch is set to 100, cross entropy is leveraged as the loss function during training and the Stochastic Gradient Descent (SGD) is utilized as the optimizer during the training process. A list of learning rate configurations (i.e., {0.001, 0.005, 0.01}) are configured. The empirical study shows that the 0.01 learning rate can bring out the best results, and here I will report the performance of my best CNN model in Figure 9 and Figure 10. Figure 9 reports the overall f1 score and micro f1 scores for each class for my CNN model on test set. Figure 10 reports the overall test accuracy and test accuracy for each class for my CNN model.



```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(5, 5), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=1024, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=1024, out_features=10, bias=True)
  )
)
```

Figure 8: The detailed structure of my CNN model

Table 5 reports the test accuracy of my CNN model trained in different learning rates.
In addition, I replaced the Relu activation function to tanh and sigmoid functions, which are the usual activation function to train a neural network. The experimental

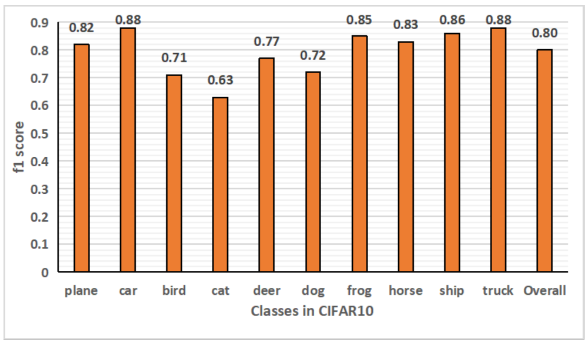


Figure 9: Overall f1 scores and f1 scores for each class for my CNN model on test set

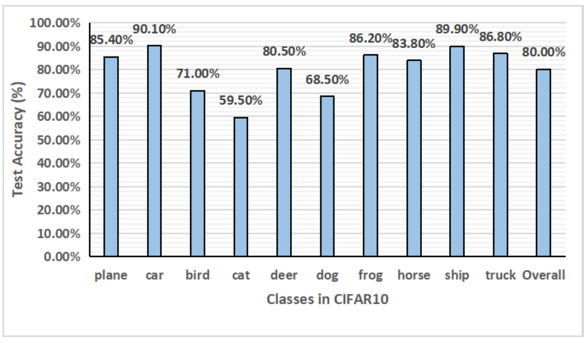


Figure 10: Overall test accuracy and test accuracy for each class for my CNN model

Table 5: The averaged test accuracy of my CNN model trained in different learning rates

Learning Rate	0.001	0.005	0.01
Test Accruacy	78.12%	79.64%	80.33%

results shows that AlexNet with these two activation functions converge slowly during training, which obtain an test accuracy of 30% and 32% respectively for tanh and sigmoid, when trained after 100 epoches. The underlying reason is the gradient decent. As tanh and sigmoid are non-linear activation functions and the slope of certain parts of the function (e.g., when the input is larger than 1) are close to zero. Therefore, the close-to-zero gradients can fail to effectively update the weights in the network and learn the characteristics from the inputs.

6 Task 4

To compare and contrast the MLP-based approach and CNN-based approach, we compare the performance of the best MLP and CNN models on test set in terms of precision, recall and f1 scores for each class in CIFAR10 dataset and their overall values. For the MLP model, the best model is with 400 hidden nodes in the hidden layer, using original data as the input and 0.001 as the learning rate. The best CNN model is the model I presented in Section 5. Both models are trained for 100 epoch for evaluation.

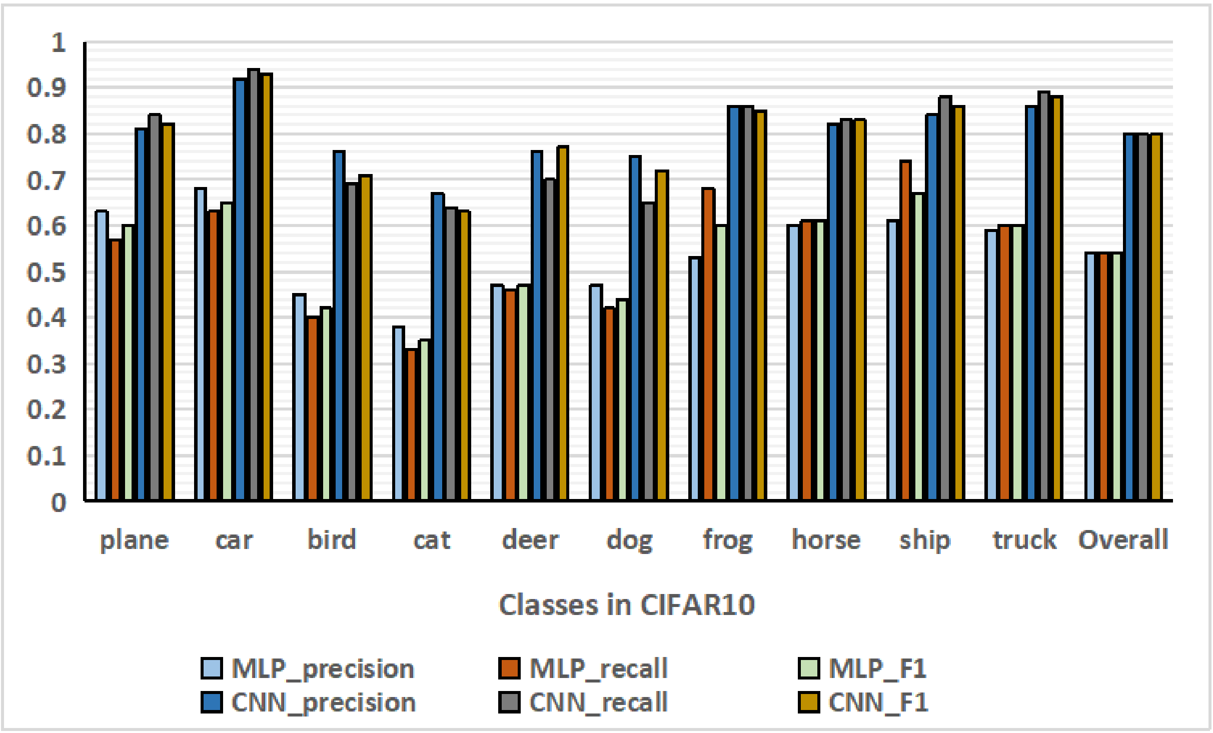


Figure 11: Best precision, recall and f1 scores for each classes and best overall precision, recall and f1 scores evaluated on CIFAR10 test set of my MLP and CNN models.

Figure 11 reports the experimental results. We make two observations. First, the CNN model outperforms the MLP model for the estimation of all the classes, in terms of precision, recall , f1 scores and estimation accuracy (presented in the above section, 78%

vs 54%), which indicates that my CNN model is a better solution for this task as MLP model is slightly better than random guessing. Second, both models achieve relatively poor estimation accuracy for *bird, cat, deer* and *dog* classes, compared with the rest of the classes, which indicates the the images of these four classes are more challenging for the current model design for learning.

In addition, we compare the time complexity of both the MLP and CNN model. For MLP model training, each epoch only takes 10 seconds while it costs 90 seconds for CNN. This is because the CNN model has deeper model structure and the convolution operations are more time-consuming. To conclude, although CNN has more complex time complexity than MLP model in my design, it significantly outperforms MLP model on object detection task on CIFAR10 dataset, which proves that it is a better solution in this case.

References

[1] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," tech. rep., University of Toronto, 01 2009.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012.*, pp. 1106–1114, 2012.