A Detailed Proofs

A.1 Proof of Lemma 1

Lemma 1. *For a graph $\mathcal{G}$ with N nodes, C classes, and N/C nodes for each class, if we randomly connect nodes to form the edge set $\mathcal{E}$, the expected homophily ratio is $\mathbb{E}(h(\mathcal{G})) = \frac{1}{C}$.*

Proof. We first randomly sample a node s from $\mathcal{V}$, assuming its label is y_s. Then we sample another node t. Because the classes are balanced, we have

$$\begin{aligned} \mathrm{P}(y_s = y_t) &= \frac{1}{C}\ , \\ \mathrm{P}(y_s \neq y_t) &= \frac{C-1}{C}\ . \end{aligned} \tag{10}$$

Therefore, if each node pair in $\mathcal{E}$ is sampled randomly, we have

$$\mathbb{E}(h) = \frac{1}{|\mathcal{E}|} \cdot |\mathcal{E}| \cdot \frac{1}{C} = \frac{1}{C}, \tag{11}$$

which completes the proof. □

A.2 Proof of Theorem 1

Theorem 1. *Let $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ be an undirected graph. $0 \leq \lambda_1 \cdots \leq \lambda_N$ are eigenvalues of its Laplacian matrix $\mathbf{L}$. Let g_1 and g_2 be two spectral filters satisfying the following two conditions: (1) $g_1(\lambda_i) < g_2(\lambda_i)$ for $1 \leq i \leq m$; and $g_1(\lambda_i) > g_2(\lambda_i)$ for $m+1 \leq i \leq N$, where $1 < m < N$; and (2) They have the same norm of output values $\|[g_1(\lambda_1), \cdots, g_1(\lambda_N)]^\top\|_2^2 = \|[g_2(\lambda_1), \cdots, g_2(\lambda_N)]^\top\|_2^2$. For a graph signal $\mathbf{x}$, $\mathbf{x}^{(1)} = g_1(\mathbf{L})\mathbf{x}$ and $\mathbf{x}^{(2)} = g_2(\mathbf{L})\mathbf{x}$ are the corresponding representations after filters g_1 and g_2. Let $\Delta s = \sum_{(s,t)\in\mathcal{E}} \left[(x_s^{(1)} - x_t^{(1)})^2 - (x_s^{(2)} - x_t^{(2)})^2 \right]$ be the difference between the total distance of connected nodes got by g_1 and g_2, where x_s^1 denotes the s-th element of $\mathbf{x}_s^{(1)}$. Then we have $\mathbb{E}[\Delta s] > 0$.*

Proof. **1.** Given the eigendecomposition of Laplacian $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$, because $\mathbf{u}_0, \cdots, \mathbf{u}_{N-1}$ are orthonormal eigenvectors, any unit graph signal $\mathbf{x}$ can be expresses as the linear combination of the eigenvectors:

$$\mathbf{x} = \sum_{i=1}^{N} c_i \mathbf{u}_i, \tag{12}$$

where $c_i = \mathbf{u}_i^T \mathbf{x}$ are the coefficients of each eigenvector. Then, we have

$$\begin{aligned} \mathbf{x}^{(1)} &= g_1(\mathbf{L})\mathbf{x} = \mathbf{U}g_1(\mathbf{\Lambda})\mathbf{U}^\top\mathbf{x} = \left(\sum_{i=1}^{N} g_1(\lambda_i)\mathbf{u}_i\mathbf{u}_i^\top\right)\left(\sum_{i=1}^{N} c_i\mathbf{u}_i\right) = \sum_{i=1}^{N} g_1(\lambda_i)c_i\mathbf{u}_i, \\ \mathbf{x}^{(2)} &= g_2(\mathbf{L})\mathbf{x} = \mathbf{U}g_2(\mathbf{\Lambda})\mathbf{U}^\top\mathbf{x} = \left(\sum_{i=1}^{N} g_2(\lambda_i)\mathbf{u}_i\mathbf{u}_i^\top\right)\left(\sum_{i=1}^{N} c_i\mathbf{u}_i\right) = \sum_{i=1}^{N} g_2(\lambda_i)c_i\mathbf{u}_i. \end{aligned} \tag{13}$$

Note that λ_i and u_i are the eigenvalues and eigenvectors of original Laplacian $\mathbf{L}$. Moreover, we have

$$\mathbf{c} = \mathbf{U}^\top\mathbf{x} \qquad c_i = \mathbf{u}_i^\top\mathbf{x}. \tag{14}$$

Eq. 14 demonstrates that each element of $\mathbf{c}$ is determined independently by the product of each eigenvalue and $\mathbf{x}$. We have $-1 = -\|\mathbf{u}_i\|_2\|\mathbf{x}\|_2 \leq c_i^2 \leq \|\mathbf{u}_i\|_2\|\mathbf{x}\|_2 = 1$. Furthermore, because x is an arbitrary unit graph signal, it can achieve any value with $\|\mathbf{x}\|_2 = 1$. It's reasonable for us to assume that c_i's are independently identically distributed with mean 0.

2. For any graph signal $\mathbf{x}$, its smoothness is the total distance between the connected nodes, which is given by,

$$\begin{aligned}\sum_{(s,t)\in\mathcal{E}} (x_s - x_t)^2 &= \mathbf{x}^\top \mathbf{L}\mathbf{x}, \\ \sum_{(s,t)\in\mathcal{E}} (x_s^{(1)} - x_t^{(2)})^2 &= \mathbf{x}^{(1)\top} \mathbf{L}\mathbf{x}^{(1)}, \\ \sum_{(s,t)\in\mathcal{E}} (x_s^{(2)} - x_t^{(2)})^2 &= \mathbf{x}^{(2)\top} \mathbf{L}\mathbf{x}^{(2)}.\end{aligned} \tag{15}$$

Note that the smoothness score of an eigenvector equals the corresponding eigenvalue:

$$\lambda_i = \mathbf{u}_i^\top \mathbf{L}\mathbf{u}_i = \sum_{(s,t)\in\mathcal{E}} (u_{i,s} - u_{i,t})^2. \tag{16}$$

Then we plug in Eq. 12 and Eq. 13 into Eq. 15 to get,

$$\sum_{(s,t)\in\mathcal{E}} (x_s - x_t)^2 = \left(\sum_{i=1}^N c_i \mathbf{u}_i^\top\right)\left(\sum_{i=1}^N \lambda_i \mathbf{u}_i \mathbf{u}_i^\top\right)\left(\sum_{i=1}^N c_i \mathbf{u}_i\right) = \sum_{i=1}^N c_i^2 \lambda_i, \tag{17}$$

$$\sum_{(s,t)\in\mathcal{E}} (x_s^{(1)} - x_t^{(1)})^2 = \left(\sum_{i=1}^N g_1(\lambda_i) c_i \mathbf{u}_i^\top\right)\left(\sum_{i=1}^N \lambda_i \mathbf{u}_i \mathbf{u}_i^\top\right)\left(\sum_{i=1}^N g_1(\lambda_i) c_i \mathbf{u}_i\right) = \sum_{i=1}^N c_i^2 \lambda_i g_1^2(\lambda_i). \tag{18}$$

$$\sum_{(s,t)\in\mathcal{E}} (x_s^{(2)} - x_t^{(2)})^2 = \left(\sum_{i=1}^N g_2(\lambda_i) c_i \mathbf{u}_i^\top\right)\left(\sum_{i=1}^N \lambda_i \mathbf{u}_i \mathbf{u}_i^\top\right)\left(\sum_{i=1}^N g_2(\lambda_i) c_i \mathbf{u}_i\right) = \sum_{i=1}^N c_i^2 \lambda_i g_2^2(\lambda_i). \tag{19}$$

3. For i.i.d. random variables c_i and any $i < j$, we have

$$\begin{aligned}&\mathbb{E}[c_i^2] = \mathbb{E}[c_j^2] \\ \Rightarrow\ &\lambda_i \mathbb{E}[c_i^2] = \mathbb{E}[\lambda_i c_i^2] \le \mathbb{E}[\lambda_j c_j^2] = \lambda_j \mathbb{E}[c_j^2]\end{aligned} \tag{20}$$

We are interested in the expected difference between the total distance of connected nodes got by g_1 and g_2. Let Δs denote difference between the total distance of connected nodes got by g_1 and g_2, i.e.,

$$\Delta s = \sum_{(s,t)\in\mathcal{E}} \left[(x_s^{(1)} - x_t^{(1)})^2 - (x_s^{(2)} - x_t^{(2)})^2\right] \tag{21}$$

Then, the expected difference between the total distance of connected nodes got by g_1 and g_2 is

$$\begin{aligned}\mathbb{E}\left[\Delta s\right] &= \mathbb{E}\left[\sum_{(s,t)\in\mathcal{E}} (x_s^{(1)} - x_t^{(1)})^2\right] - \mathbb{E}\left[\sum_{(s,t)\in\mathcal{E}} (x_s^{(2)} - x_t^{(2)})^2\right] \\ &= \mathbb{E}\left[\sum_{i=1}^N c_i^2 \lambda_i g_1^2(\lambda_i)\right] - \mathbb{E}\left[\sum_{i=1}^N c_i^2 \lambda_i g_2^2(\lambda_i)\right] \\ &= \sum_{i=1}^N \left\{\left[g_1^2(\lambda_i) - g_2^2(\lambda_i)\right] \lambda_i \mathbb{E}\left[c_i^2\right]\right\}\end{aligned} \tag{22}$$

4. We assume $\|[g_1(\lambda_1), \cdots, g_1(\lambda_N)]^\top\|_2^2 = \|[g_2(\lambda_1), \cdots, g_2(\lambda_N)]^\top\|_2^2$ so that g_1 and g_2 have the same ℓ_2-norms. We make this assumption to avoid some trivial solutions. For example, if we simply multiply the representation $\mathbf{x}$ with a constant, the value in Eq. 17 will also be enlarged and reduced, but the discriminative ability is unchanged. Therefore, we have

$$\begin{aligned}
&\sum_{i=1}^{N} g_1^2(\lambda_i) = \sum_{i=1}^{N} g_2^2(\lambda_i) \\
\Rightarrow &\sum_{i=1}^{m} g_1^2(\lambda_i) + \sum_{i=m+1}^{N} g_1^2(\lambda_i) = \sum_{i=1}^{m} g_2^2(\lambda_i) + \sum_{i=m+1}^{N} g_2^2(\lambda_i) \\
\Rightarrow &\, 0 < \sum_{i=1}^{m} \left[g_2^2(\lambda_i) - g_1^2(\lambda_i)\right] = \sum_{i=m+1}^{N} \left[g_1^2(\lambda_i) - g_2^2(\lambda_i)\right],
\end{aligned} \tag{23}$$

because $g_1(\lambda_i) < g_2(\lambda_i)$ for $1 \leq i \leq m$ and $g_1(\lambda_i) > g_2(\lambda_i)$ for $m+1 \leq i \leq N$. By applying the results in Eq. 23 and Eq. 20, we have

$$\begin{aligned}
0 &< \sum_{i=1}^{m} \left\{\left[g_2^2(\lambda_i) - g_1^2(\lambda_i)\right]\lambda_i\right\} < \lambda_m \sum_{i=1}^{m} \left[g_2^2(\lambda_i) - g_1^2(\lambda_i)\right] \\
&< \lambda_{m+1} \sum_{i=1}^{m} \left[g_2^2(\lambda_i) - g_1^2(\lambda_i)\right] = \lambda_{m+1} \sum_{i=m+1}^{N} \left[g_1^2(\lambda_i) - g_2^2(\lambda_i)\right] \\
&< \sum_{i=m+1}^{N} \left\{\left[g_1^2(\lambda_i) - g_2^2(\lambda_i)\right]\lambda_i\right\}.
\end{aligned} \tag{24}$$

Then we can derive that

$$\begin{aligned}
&\sum_{i=1}^{N} \left\{\left[g_1^2(\lambda_i) - g_2^2(\lambda_i)\right]\lambda_i\right\} > 0 \\
\Rightarrow &\sum_{i=1}^{N} \left\{\left[g_1^2(\lambda_i) - g_2^2(\lambda_i)\right]\lambda_i \mathbb{E}\left[c_i^2\right]\right\} = \mathbb{E}\left[\Delta s\right] > 0 \\
\Rightarrow &\mathbb{E}\big[\sum_{(s,t)\in\mathcal{E}} (x_s^{(1)} - x_t^{(1)})^2\big] > \mathbb{E}\big[\sum_{(s,t)\in\mathcal{E}} (x_s^{(2)} - x_t^{(2)})^2\big]
\end{aligned} \tag{25}$$

which completes our proof. □

A.3 Proof of Theorem 2

Theorem 2. *Let $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ be a balanced undirected graph with N nodes, C classes, and N/C nodes for each class. $\mathcal{P}_{in}$ is the set of possible pairs of nodes from the same class. $\mathcal{P}_{out}$ is the set of possible pairs of nodes from different classes. g_1 and g_2 are two filters same as Theorem 1. Given an arbitrary graph signal $\mathbf{x}$, let $d_{in}^{(1)} = \sum_{(s,t)\in\mathcal{P}_{in}} \left(x_s^{(1)} - x_t^{(1)}\right)^2$ be the total intra-class distance, $d_{out}^{(1)} = \sum_{(s,t)\in\mathcal{P}_{out}} \left(x_s^{(1)} - x_t^{(1)}\right)^2$ be the total inter-class distance, and $\bar{d}_{in}^{(1)} = d_{in}^{(1)}/|\mathcal{P}_{in}|$ be the average intra-class distance while $\bar{d}_{out}^{(1)} = d_{out}^{(1)}/|\mathcal{P}_{out}|$ be the average inter-class distance. $\Delta\bar{d}^{(1)} = \bar{d}_{out}^{(1)} - \bar{d}_{in}^{(1)}$ is the difference between average inter-distance and intra-class distance. $d_{out}^{(2)}$, $d_{out}^{(2)}$, $\bar{d}_{in}^{(2)}$, $\bar{d}_{out}^{(2)}$, and $\Delta\bar{d}^{(2)}$ are corresponding values defined similarly on $\mathbf{x}^{(2)}$. If $\mathbb{E}[\Delta s] > 0$, we have: (1) when $h > \frac{1}{C}$, $\mathbb{E}[\Delta\bar{d}^{(1)}] < \mathbb{E}[\Delta\bar{d}^{(2)}]$; and (2) when $h < \frac{1}{C}$, $\mathbb{E}[\Delta\bar{d}^{(1)}] > \mathbb{E}[\Delta\bar{d}^{(2)}]$.*

Proof. In graph $\mathcal{G}$, the number of possible homophilous (intra-class) edges (including self-loop) is,

$$|\mathcal{P}_{in}| = \frac{C}{2}\frac{N}{C}\left(\frac{N}{C}\right) = \frac{N^2}{2C}. \tag{26}$$

The number of possible heterophilous (inter-class) edges is,

$$|\mathcal{P}_{out}| = \frac{C}{2}\frac{N}{C}\left(\frac{C-1}{C}N\right) = \frac{N^2}{2}\left(\frac{C-1}{C}\right). \tag{27}$$

Therefore, we have

$$\bar{d}_{in}^{(i)} = \frac{d_{in}^{(i)}}{|\mathcal{P}_{in}|} = \frac{2Cd_{in}^{(i)}}{N^2}\ , \quad i \in \{1,2\} \tag{28}$$

$$\bar{d}_{out}^{(i)} = \frac{d_{out}^{(i)}}{|\mathcal{P}_{out}|} = \frac{2Cd_{out}^{(i)}}{N^2(C-1)}\ , \quad i \in \{1,2\} \tag{29}$$

$$\begin{aligned} \Delta\bar{d}^{(i)} &= \bar{d}_{out}^{(i)} - \bar{d}_{in}^{(i)} \\ &= \frac{2Cd_{out}^{(i)}}{N^2(C-1)} - \frac{2Cd_{in}^{(i)}}{N^2} \\ &= \frac{2C}{(C-1)N^2}\left[d_{out}^{(i)} - (C-1)d_{in}^{(i)}\right] \end{aligned} \tag{30}$$

$\mathcal{E}_{in}$ denotes the set of edges connecting nodes from the same class (intra-class edges). $\mathcal{E}_{out}$ denotes the set of edges connecting nodes from different classes (inter-class edges). There are $|\mathcal{E}|$ edges, $h \cdot |\mathcal{E}|$ homophilous edges, and $(1-h) \cdot |\mathcal{E}|$ heterophilous edges. In expectation, each edge has the same difference of the distance of connected nodes got by g_1 and g_2, i.e.,

$$\begin{aligned} \mathbb{E}\left[\Delta s\right] &= \mathbb{E}\left[\sum_{(s,t)\in\mathcal{E}}\left[(x_s^{(1)} - x_t^{(1)})^2 - (x_s^{(2)} - x_t^{(2)})^2\right]\right] \\ \mathbb{E}\left[h\Delta s\right] &= \mathbb{E}\left[\sum_{\substack{(s,t)\in\mathcal{E} \\ y_s = y_t}}\left[(x_s^{(1)} - x_t^{(1)})^2 - (x_s^{(2)} - x_t^{(2)})^2\right]\right] \\ \mathbb{E}\left[(1-h)\Delta s\right] &= \mathbb{E}\left[\sum_{\substack{(s,t)\in\mathcal{E} \\ y_s \neq y_t}}\left[(x_s^{(1)} - x_t^{(1)})^2 - (x_s^{(2)} - x_t^{(2)})^2\right]\right] \end{aligned} \tag{31}$$

If we solely consider the direct influence of graph convolution on connected nodes, the relationship between d' and d can be expressed as follows:

$$\mathbb{E}\left[\bar{d}_{out}^{(1)} - \bar{d}_{out}^{(2)}\right] = (1-h)\mathbb{E}\left[\Delta s\right] \quad \text{and} \quad \mathbb{E}\left[\bar{d}_{in}^{(1)} - \bar{d}_{in}^{(2)}\right] = h\mathbb{E}\left[\Delta s\right] \tag{32}$$

$$\begin{aligned} \mathbb{E}\left[\Delta\bar{d}^{(1)} - \Delta\bar{d}^{(2)}\right] &= \mathbb{E}\left[\frac{2C}{(C-1)N^2}\left[(d_{out}^{(1)} - d_{out}^{(2)}) - (C-1)(d_{in}^{(1)} - d_{in}^{(2)})\right]\right] \\ &= \frac{2C}{(C-1)N^2}\left[(1-h)\mathbb{E}[\Delta s] - (C-1)h\mathbb{E}[\Delta s]\right] \\ &= \frac{2C}{(C-1)N^2}\left[\mathbb{E}[\Delta s](1-Ch)\right] \end{aligned} \tag{33}$$

Because $\frac{2C}{(C-1)N^2} > 0$, then we have
(1) when $h > \frac{1}{C}$, if $\Delta s < 0$, then $\mathbb{E}[\Delta\bar{d}^{(1)}] > \mathbb{E}[\Delta\bar{d}^{(2)}]$; if $\Delta s > 0$, then $\mathbb{E}[\Delta\bar{d}^{(1)}] < \mathbb{E}[\Delta\bar{d}^{(2)}]$.
(2) when $h < \frac{1}{C}$, if $\Delta s < 0$, then $\mathbb{E}[\Delta\bar{d}^{(1)}] < \mathbb{E}[\Delta\bar{d}^{(2)}]$; if $\Delta s > 0$, then $\mathbb{E}\Delta\bar{d}^{(1)}] > \mathbb{E}[\Delta\bar{d}^{(2)}]$. □

B TRAINING ALGORITHM OF NEWTONNET

We show the training algorithm of NewtonNet in Algorithm 1. We first randomly initialize θ, h, and $\{t_0, \cdots, t_K\}$. We update h according to the predicted labels of each iteration. We update the learned representations and θ, and $\{t_0, \cdots, t_K\}$ accordingly until convergence or reaching max iteration.

Algorithm 1 Training Algorithm of NewtonNet

```
Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, $\mathcal{Y}_L$, $K$, $\{q_0, \cdots, q_K\}$, $\gamma_1, \gamma_2, \gamma_3$,
Output: $\theta$, $h$, $\{t_0, \cdots, t_K\}$
1: Randomly initialize $\theta$, $h$, $\{t_0, \cdots, t_K\}$
2: repeat
3:     Update representations $\mathbf{z}$ by Eq. 6
4:     Update predicted labels $\hat{\boldsymbol{y}}_v$
5:     Update $h$ with predicted labels
6:     $\mathcal{L} \leftarrow$ Eq. 9
7:     Update $\theta$ and $\{t_0, \dots, t_K\}$
8: until convergence or reaching max iteration
```

B.1 Complexity Analysis

Time complexity. According to Blakely et al. (2021), the time complexity of GCN is $O(L(EF + NF^2))$, where E is the number of edges, and L is the number of layers. One-layer GCN has the formula $\mathbf{X}^{l+1} = \sigma(\mathbf{A}\mathbf{X}^l\mathbf{W}^l)$. $O(NF^2)$ is the time complexity of feature transformation, and $O(EF)$ is the time complexity of neighborhood aggregation. Following GPRGNN, ChebNetII, and BernNet, NewtonNet uses the "first transforms features and then propagates" strategy. According to Eq. 6, the feature transformation part is implemented by an MLP with time $O(NF^2)$. And the propagation part has the time complexity with $O(KEF)$. In other words, NewtonNet has a time complexity $O(KEF + NF^2)$, which is linear to K. BernNet's time complexity is quadratic to K. We summarize the time complexity in Table 2.

Table 2: Time and space complexity.

Method	Time Complexity	Space Complexity
MLP	$O(NF^2)$	$O(NF^2)$
GCN	$O(L(EF + NF^2))$	$O(E + LF^2 + LNF)$
GPRGNN	$O(KEF + NF^2)$	$O(E + F^2 + NF)$
ChebNetII	$O(KEF + NF^2)$	$O(E + F^2 + NF)$
BernNet	$O(K^2EF + NF^2)$	$O(KE + F^2 + NF)$
NewtonNet	$O(KEF + NF^2)$	$O(KE + F^2 + NF)$

Table 3: Running Time (ms/epoch) of each method.

Method	Chameleon	Pubmed	Penn94	Genius
MLP	1.909	2.283	6.119	10.474
GCN	2.891	3.169	22.043	20.159
Mixhop	3.609	4.299	19.702	27.041
GPRGNN	4.807	4.984	10.572	12.522
ChebNetII (K=5)	4.414	4.871	9.609	12.189
ChebNetII (K=10)	7.352	7.447	13.661	15.346
BernNet (K=5)	8.029	11.730	19.719	20.168
BernNet (K=10)	20.490	20.869	49.592	43.524
NewtonNet (K=5)	6.6135	7.075	17.387	17.571
NewtonNet (K=10)	12.362	13.042	25.194	30.836

To examine our analysis, Table 3 shows the running time of each method. We employ 5 different masks with 2000 epochs and calculate the average time of each epoch. We observe that (1) For the spectral methods, NewtonNet, ChebNetII, and GPRGNN run more quickly than BernNet since their time complexity is linear to K while BernNet is quadratic; (2) NewtonNet cost more time than GPRGNN and ChebNetII because it calculates a more complex polynomial; (3) On smaller datasets (Chameleon, Pubmed), GCN runs faster than NewtonNet. On larger datasets (Penn94, Genius), NewtonNet and other spectral methods are much more efficient than GCN. This is because we only transform and propagate the features once, but in GCN, we stack several layers to propagate to more neighbors. In conclusion, NewtonNet is scalable on large datasets.

Space complexity. Similarly, GCN has a space complexity $O(E + LF^2 + LNF)$. $O(F^2)$ is for the weight matrix of feature transformation while $O(NF)$ is for the feature matrix. $O(E)$ is caused by the sparse edge matrix. NewtonNet has the space complexity $O(KE + F^2 + NF)$ because it pre-calculates $(\mathbf{L} - q_i\mathbf{I})$ in Equation 6. We compare the space complexity in Table 2 and the memory used by each model in Table 4. On smaller datasets, NewtonNet has a similar space consumption with GCN. However, NewtonNet and other spectral methods are more space efficient than GCN on larger datasets because we do not need to stack layers. Therefore, NewtonNet has excellent scalability.

Table 4: Memory usage (MB) of each method.

Method	Chameleon	Pubmed	Penn94	Genius
MLP	1024	1058	1862	1390
GCN	1060	1114	3320	2012
Mixhop	1052	1124	2102	2536
GPRGNN	1046	1060	1984	1370
ChebNetII	1046	1080	1982	1474
BernNet	1046	1082	2026	1544
NewtonNet	1048	1084	2292	1868

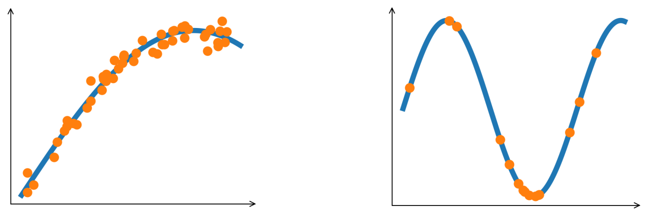

(a) Approximation. (b) Interpolation.

Figure 6: Difference between approximation and interpolation.

C Approximation vs Interpolation

Approximation and interpolation are two common curve fitting methods (Hildebrand, 1987). Given an unknown continuous function $\hat{f}(x)$, and its values at $n + 1$ known points $\{(x_0, \hat{f}(x_0)), \cdots, (x_n, \hat{f}(x_n))\}$, we want to use $g(x)$ to fit unknown $\hat{f}(x)$. Approximation methods aim to minimize the error between the original function and estimated function $|\hat{f}(x) - g(x)|$; while interpolation methods aim to fit the data and make $\hat{f}(x_i) = g(x_i), i = 0, \cdots, n$. In other words, the interpolation function passes every known point exactly, but the approximation function finds minimal error among the known points. Fig. 6 shows the difference. The property of interpolation allows us to learn the function values directly in our model, which is discussed in Section 4.

D Experimental Details

D.1 Synthetic Datasets

In this paper, we employ contextual stochastic block model (CSBM) (Deshpande et al., 2018) to generate synthetic datasets. CSBM provides a model to generate synthetic graphs with controllable inter-class and intra-class edge probability. It assumes features of nodes from the same class conform to the same distribution. Assume there are two equal-size classes $\{c_0, c_1\}$ with n nodes for each class. CSBM generates edges and features by:

$$\mathbb{P}(\mathbf{A}_{ij} = 1) = \begin{cases} \frac{1}{n}(d + \sigma\sqrt{d}) & \text{when } y_i = y_j \\ \frac{1}{n}(d - \sigma\sqrt{d}) & \text{when } y_i \neq y_j \end{cases} \qquad \mathbf{x}_i = \sqrt{\frac{\mu}{n}} v_i \mathbf{u} + \frac{\mathbf{w}_i}{\sqrt{F}} \tag{34}$$

where d is the average node degree, μ is mean value of Gaussian distribution, F is the feature dimension, entries of $\mathbf{w}_i \in \mathbb{R}^p$ has independent standard normal distributions, and $\mathbf{u} \sim \mathcal{N}(0, I_F/F)$. We can control homophily ratio h by changing $\sigma = \sqrt{d}(2h-1)$, $-\sqrt{d} \leq \sigma \leq \sqrt{d}$. When $\sigma = -\sqrt{d}$, it is a totally heterophilous graph; when $\sigma = \sqrt{d}$, it is a totally homophilous graph. Following (Chien et al., 2021), we adopt $d = 5, \mu = 1$ in this paper. We vary σ to generate graphs with different homophily levels. In Fig. 1(b), we adopt $2n = 3000, F = 3000$ to generate the synthetic dataset. We vary the number of nodes $2n$ and number of features F to generate different CSBM datasets and show their frequency importance in Fig. 9.

D.2 Real-world datasets

Citation Networks (Sen et al., 2008): Cora, Citeseer, and PubMed are citation network datasets. Cora consists of seven classes of machine learning papers, while CiteSeer has six. Papers are represented by nodes, while citations between two papers are represented by edges. Each node has features defined by the words that appear in the paper's abstract. Similarly, PubMed is a collection of abstracts from three types of medical papers.

WebKB (Pei et al., 2020): Cornell, Texas, and Wisconsin are three sub-datasets of WebKB. They are collected from a set of websites of several universities' CS departments and processed by (Pei et al., 2020). For each dataset, a node represents a web page and an edge represents a hyperlink. Node features are the bag-of-words representation of each web page. We aim to classify the nodes into one of the five classes, student, project, course, staff, and faculty.

Wikipedia Networks (Rozemberczki et al., 2021): Chameleon, Squirrel, and Crocodile are three topics of Wikipedia page-to-page networks. Articles from Wikipedia are represented by nodes, and links between them are represented by edges. Node features indicate the occurrences of specific nouns in the articles. Based on the average monthly traffic of the web page, the nodes are divided into five classes.

Social Networks (Lim et al., 2021): Penn94 (Traud et al., 2012) is a social network of friends among university students on Facebook in 2005. The network consists of nodes representing individual students, each with their reported gender identified. Additional characteristics of the nodes include their major, secondary major/minor, dormitory or house, year of study, and high school attended.

Twitch-gamers (Rozemberczki & Sarkar, 2021) is a network graph of Twitch accounts and their mutual followers. Node attributes include views, creation date, language, and account status. The classification is binary and to predict whether the channel has explicit content.

Genius (Lim & Benson, 2021) is from the genius.com social network, where nodes represent users and edges connect mutual followers. Node attributes include expertise scores, contribution counts, and user roles. Some users are labeled "gone," often indicating spam. Our task is to predict these marked nodes.

Table 5: Statistics of real-world datasets.

Dataset	**Citation**			**Wikipedia**			**WebKB**		**Social**		
	Cora	Cite.	Pubm.	Cham.	Squi.	Croc.	Texas	Corn.	Penn94	Genius	Gamer
Nodes	2708	3327	19717	2277	5201	11,631	183	183	41554	421,961	168,114
Edges	5429	4732	44338	36101	217,073	360040	309	295	1,362,229	984,979	6,797,557
Attributes	1433	3703	500	2325	2089	128	1703	1703	4814	12	7
Classes	7	6	3	5	5	5	5	5	2	2	2
h	0.81	0.74	0.80	0.24	0.22	0.25	0.11	0.31	0.47	0.62	0.55

D.3 Baselines

We compare our method with various state-of-the-art methods for both spatial and spectral methods. First, we compare the following spatial methods:

- **MLP**: Multilayer Perceptron predicts node labels using node attributes only without incorporating graph structure information.
- **GCN** (Kipf & Welling, 2017): Graph Convolutional Network is one of the most popular MPNNs using 1-hop neighbors in each layer.

- **MixHop** (Abu-El-Haija et al., 2019): MixHop mixes 1-hop and 2-hop neighbors to learn higher-order information.
- **APPNP** (Klicpera et al., 2018): APPNP uses the Personalized PageRank algorithm to propagate the prediction results of GNN to increase the propagation range.
- **GloGNN++** (Li et al., 2022): GloGNN++ is a method for creating node embeddings by aggregating information from global nodes in a graph using coefficient matrices derived through optimization problems.

We also compare with recent state-of-the-art spectral methods:

- **ChebNet** (Defferrard et al., 2016): ChebNet uses Chebyshev polynomial to approximate the filter function. It is a more generalized form of GCN.
- **GPRGNN** (Chien et al., 2021): GPRGNN uses Generalized PageRank to learn weights for combining intermediate results.
- **BernNet** (He et al., 2021b): ChebNet uses Bernstein polynomial to approximate the filter function. It can learn arbitrary target functions.
- **ChebNetII** (He et al., 2022): ChebNet uses Chebyshev interpolation to approximate the filter function. It mitigates the Runge phenomenon and ensures the learned filter has a better shape.
- **JacobiConv** (Wang & Zhang, 2022): JacobiConv uses Jacobi basis to study the expressive power of spectral GNNs.

D.4 Settings

We run all of the experiments with 10 random splits and report the average performance with the standard deviation. For full-supervised learning, we use 60%/20%/20% splits for the train/validation/test set. For a fair comparison, for each method, we select the best configuration of hyperparameters using the validation set and report the mean accuracy and variance of 10 random splits on the test. For NewtonNet, we choose $K = 5$ and use a MLP with two layers and 64 hidden units for encoder f_θ. We search the learning rate of encoder and propagation among $\{0.05, 0.01, 0.005\}$, the dropout rate for encoder and propagation among $\{0, 0.0005\}$, and $\gamma_1, \gamma_2, \gamma_3$ among $\{0, 1, 3, 5\}$. For other baselines, we use the original code and optimal hyperparameters from authors if available. Otherwise, we search the hyperparameters within the same search space of NewtonNet.

E Additional Experimental Results

E.1 Hyperparameter Analysis

In our hyperparameter sensitivity analysis on the Citeseer and Chameleon datasets, we investigated the effects of varying the values of γ_1, γ_2, and γ_3 among$\{0, 0.01, 0.1, 1, 10, 100\}$. The accuracy results were plotted in Figure 7. We made the following observations. For the Chameleon dataset, increasing the value of γ_1 resulted in improved performance, as it effectively discouraged low-frequency components. As for γ_2, an initial increase led to performance improvements since it balanced lower and higher frequencies. However, further increases in γ_2 eventually led to a decline in performance. On the other hand, increasing γ_3 had a positive effect on performance, as it encouraged the inclusion of more high-frequency components.

Regarding the Citeseer dataset, we found that increasing the values of γ_1, γ_2, and γ_3 initially improved performance. However, there was a point where further increases in these regularization terms caused a decrease in performance. This can be attributed to the fact that excessively large regularization terms overshadowed the impact of the cross entropy loss, thus hindering the model's ability to learn effectively. We also observe that the change of Chameleon is more than that in Citeseer, so heterophilous graphs need more regularization.

We also investigate the sensitivity of the parameter K. While keeping the remaining optimal hyperparameters fixed, we explore different values of K within the set $\{2, 5, 8, 11, 14\}$. The corresponding accuracy results are presented in Fig. 8. In both datasets, we observe a pattern of increasing performance followed by a decline. This behavior can be attributed to the choice of K.

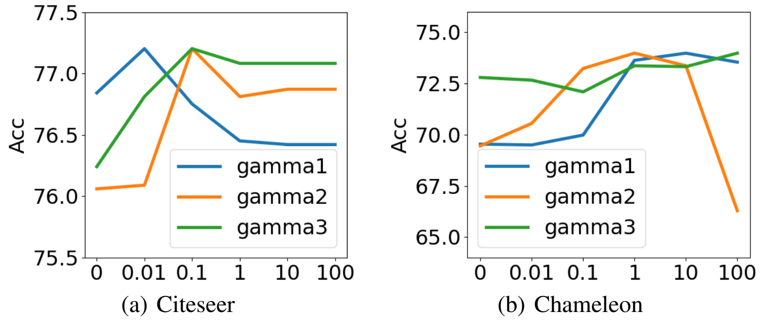


Figure 7: Hyperparameter Analysis for γ_1, γ_2, and γ_3

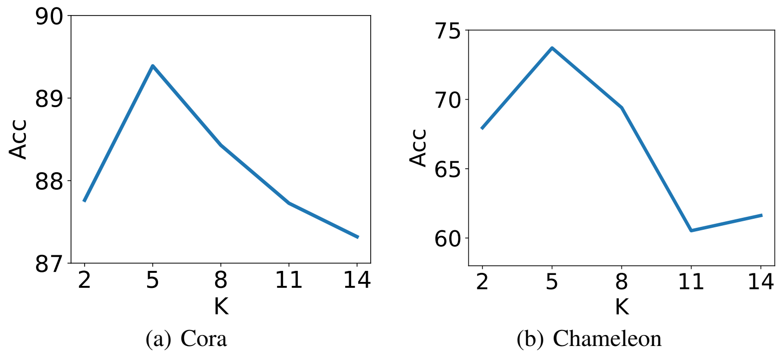


Figure 8: Hyperparameter Analysis for K.

If K is set to a small value, the polynomial lacks the necessary power to accurately approximate arbitrary functions. Conversely, if K is set to a large value, the polynomial becomes susceptible to the Runge phenomenon (He et al., 2022).

E.2 Learned Homophily Ratio

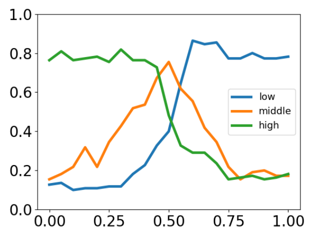
Table 6 presents the real homophily ratio alongside the learned homophily ratio for each dataset. The close proximity between the learned and real homophily ratios indicates that our model can estimate the homophily ratio accurately so that it can further guide the learning of spectral filter.

E.3 More results of frequency importance

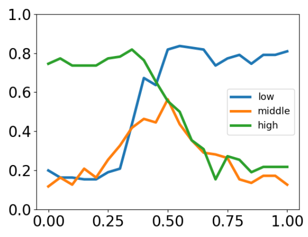
In Fig. 9, we present more results of frequency importance on CSBM datasets with different numbers of nodes and features. We fix $d = 5$ and $\mu = 1$ in Eq. 34 and vary the number of nodes and features among $\{800, 1000, 1500, 2000, 3000\}$. We can get similar conclusions as in Section 3.2.

E.4 Learned filters

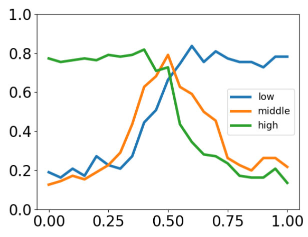
The learned filters of NewtonNet, BernNet, and ChebNetII for each dataset are illustrated in Fig. 10 to Fig. 18. Our observations reveal that NewtonNet exhibits a distinct ability to promote or discourage specific frequencies based on the homophily ratio. In the case of homophilous datasets, NewtonNet emphasizes low-frequency components while suppressing middle and high frequencies. Conversely, for heterophilous datasets, the learned spectral filter of NewtonNet emphasis more on high-frequency components compared to other models.



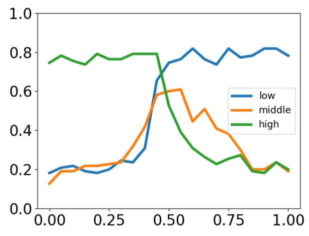
(a) 800 Nodes, 800 Features



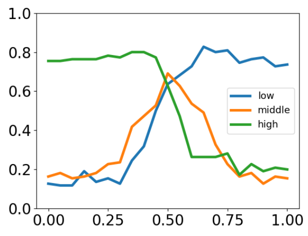
(b) 800 Nodes, 1000 Features



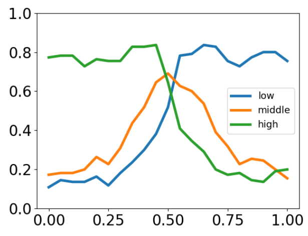
(c) 800 Nodes, 1500 Features



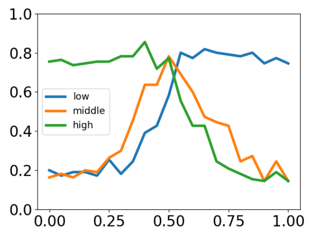
(d) 800 Nodes, 2000 Features



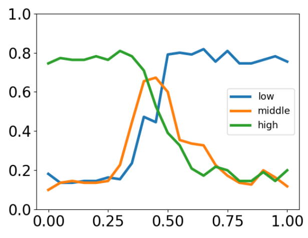
(e) 1000 Nodes, 1000 Features



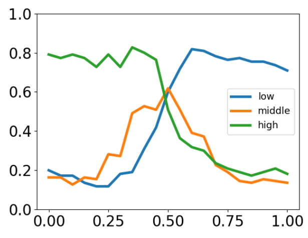
(f) 1000 Nodes, 2000 Features



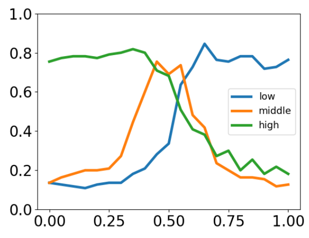
(g) 1000 Nodes, 3000 Features



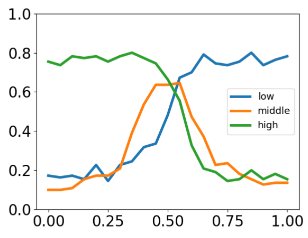
(h) 1500 Nodes, 800 Features



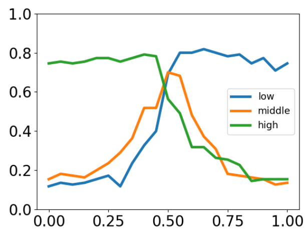
(i) 1500 Nodes, 1500 Features



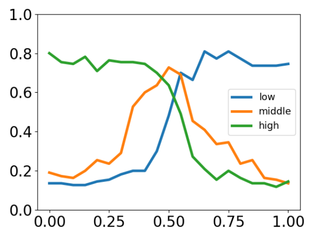
(j) 2000 Nodes, 1000 Features



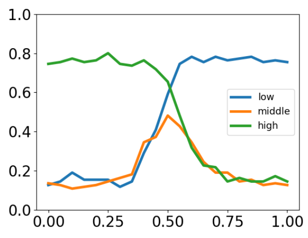
(k) 2000 Nodes, 1500 Features



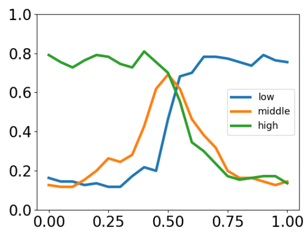
(l) 2000 Nodes, 2000 Features



(m) 2000 Nodes, 3000 Features



(n) 3000 Nodes, 1000 Features

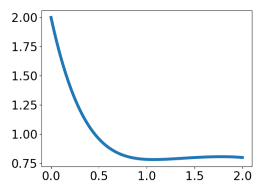


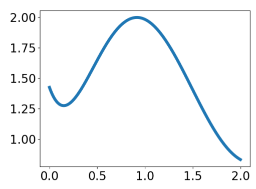
(o) 3000 Nodes, 2000 Features

Figure 9: Frequency importance on CSBM model with different hyperparameters.

Table 6: The real homophily ratio and learned homophily ratio in Table 1

	Cora	Cite.	Pubm.	Cham.	Squi.	Croc.	Texas	Corn.	Penn.
Real	0.81	0.74	0.80	0.24	0.22	0.25	0.11	0.20	0.47
Learned	0.83	0.79	0.83	0.27	0.23	0.28	0.12	0.33	0.51


(a) NewtonNet


(b) ChebNetII

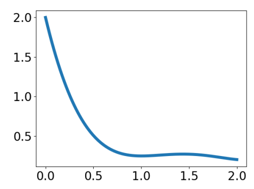
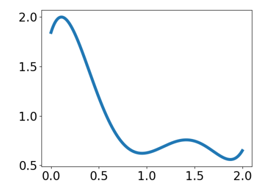
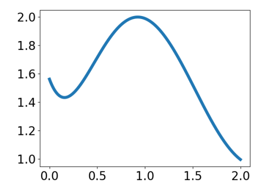

(c) BernNet

Figure 10: Learned filters on Cora.


(a) NewtonNet


(b) ChebNetII

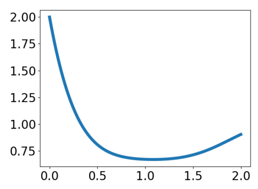
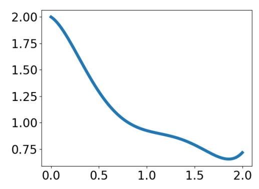
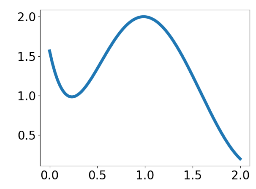

(c) BernNet

Figure 11: Learned filters on Citeseer.


(a) NewtonNet


(b) ChebNetII

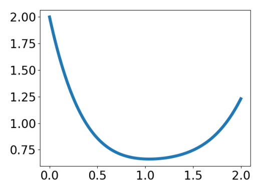
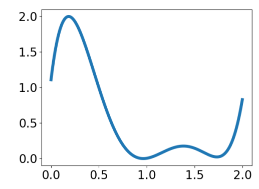
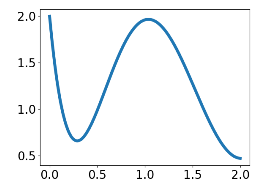

(c) BernNet

Figure 12: Learned filters on Pubmed.


(a) NewtonNet


(b) ChebNetII

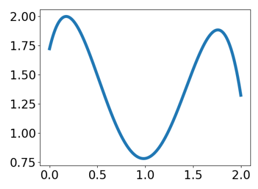
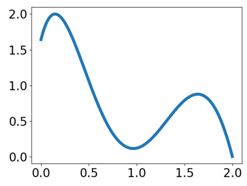
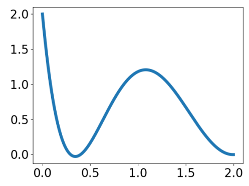

(c) BernNet

Figure 13: Learned filters on Chameleon.


(a) NewtonNet


(b) ChebNetII

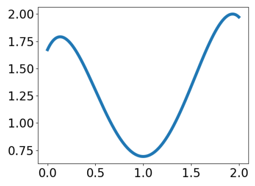
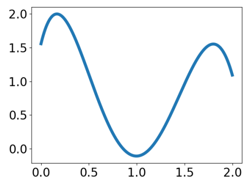
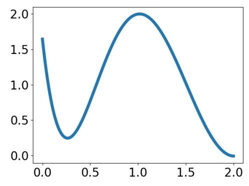

(c) BernNet

Figure 14: Learned filters on Squirrel.


(a) NewtonNet


(b) ChebNetII

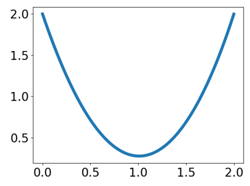
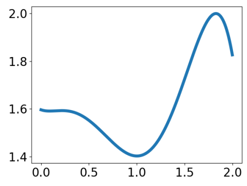
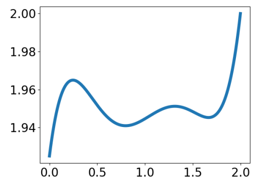

(c) BernNet

Figure 15: Learned filters on Crocodile.


(a) NewtonNet


(b) ChebNetII

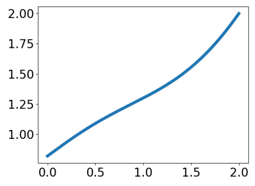
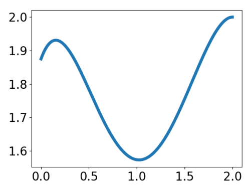
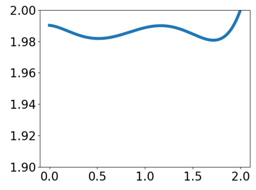

(c) BernNet

Figure 16: Learned filters on Texas.


(a) NewtonNet


(b) ChebNetII

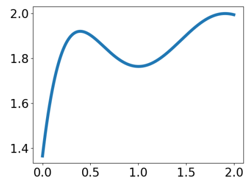
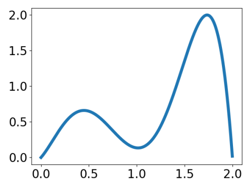
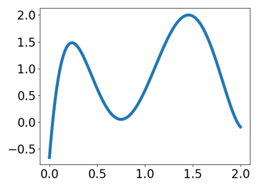

(c) BernNet

Figure 17: Learned filters on Cornell.


(a) NewtonNet


(b) ChebNetII

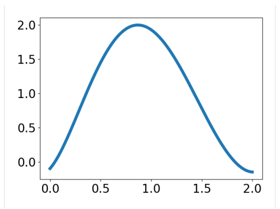

(c) BernNet

Figure 18: Learned filters on Penn94.