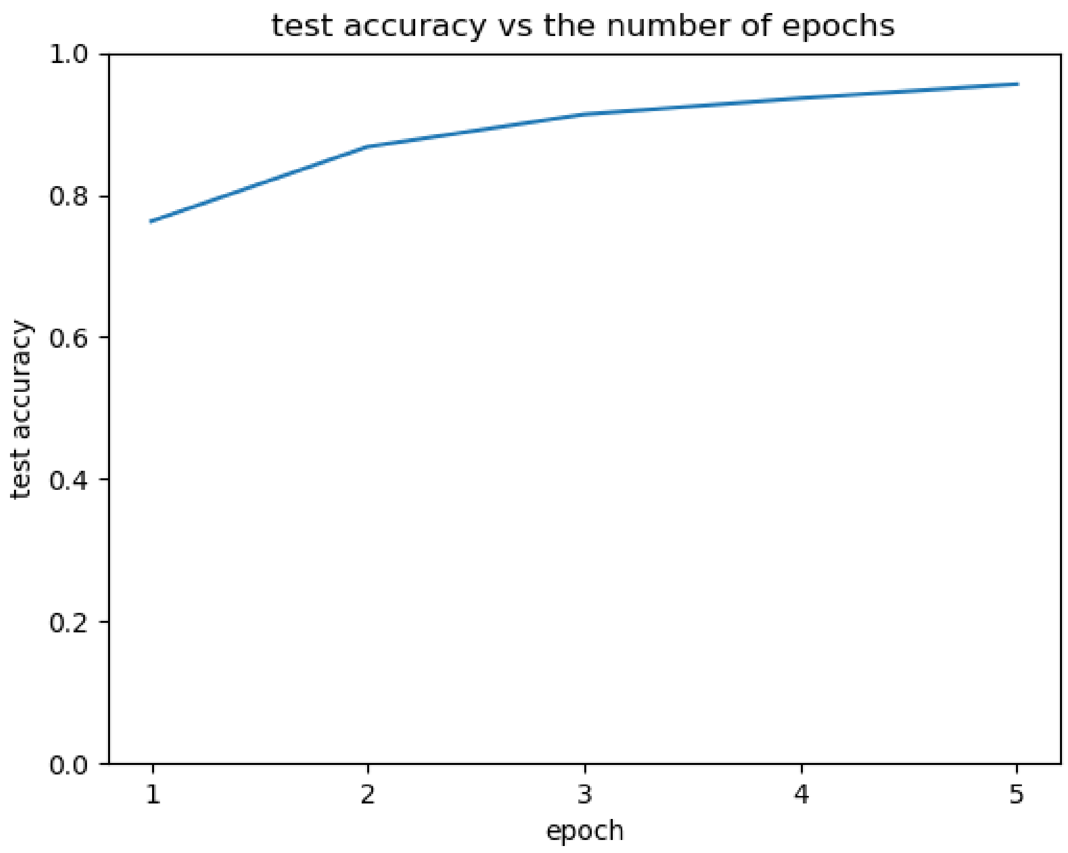
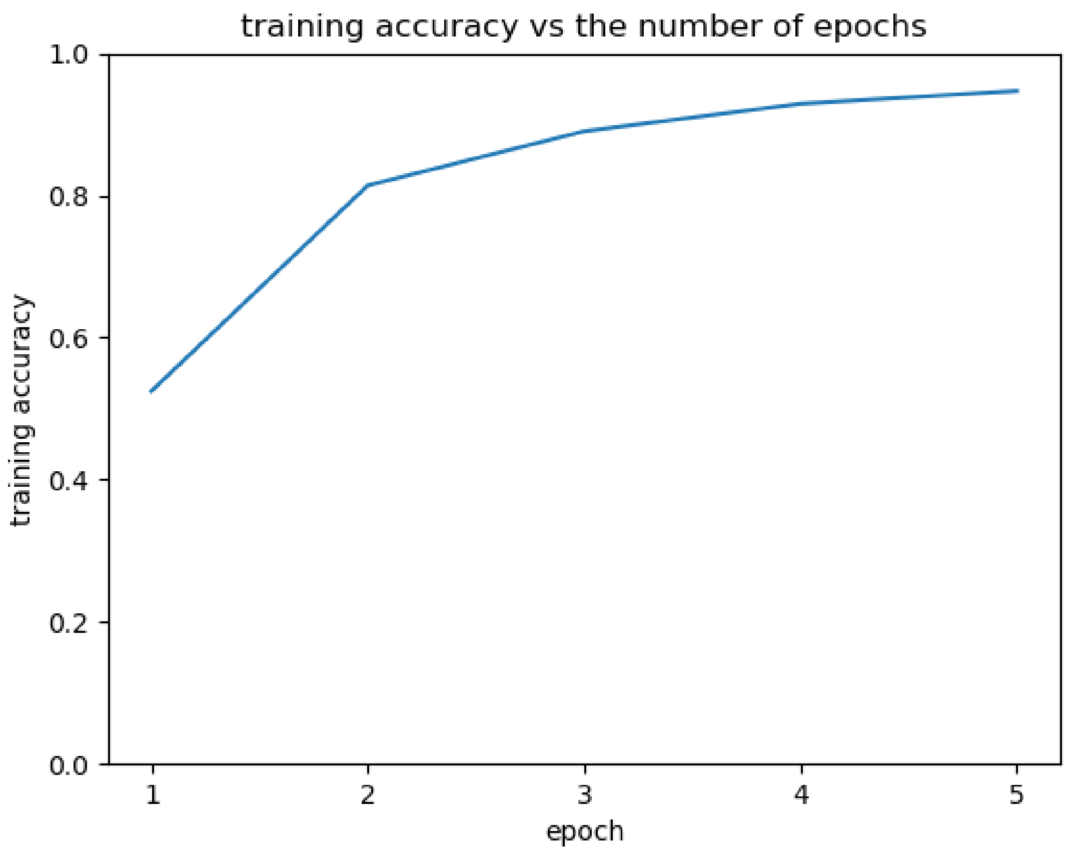
Boshen Cui 20613736 CS480-Sec001 Assignment 4

Exercise 1:

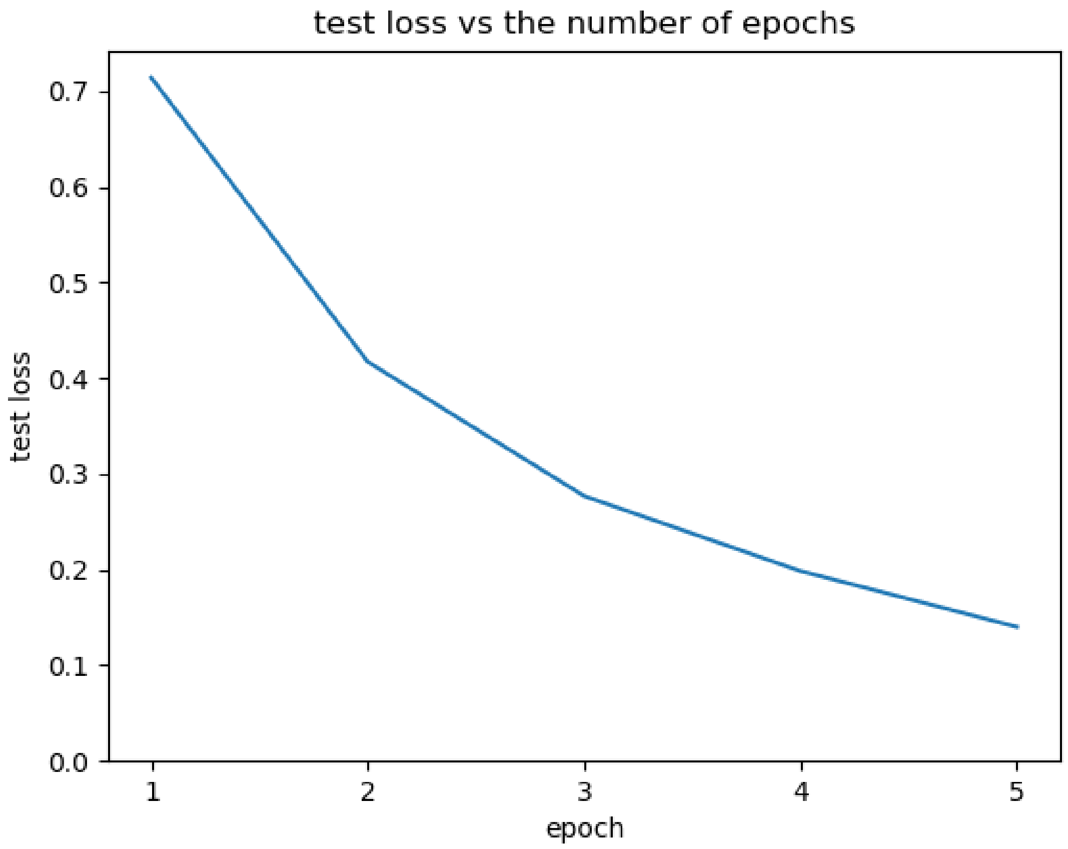
1. Code can be found in **VGG11.py**. Ensure that you have **python 3, keras, tensorflow, numpy**, and **scipy** installed. To run, simply run **python VGG11.py** (or **python3 VGG11.py** if python is binded to python2).
   Due to the long running time for training, I recommend you set up a Google Colab and run it there, with GPU acceleration.
   Alternatively, you can simply run my Google Colab, which has the exact same code:
   https://colab.research.google.com/drive/1ZtJcTOmnu_N5bx3K4e8nydc_Fo8rvz6x
   Make sure to restart the runtime every time after running to clear RAM/maintain reproducibility/prevent wonky behavior
2.
   a.



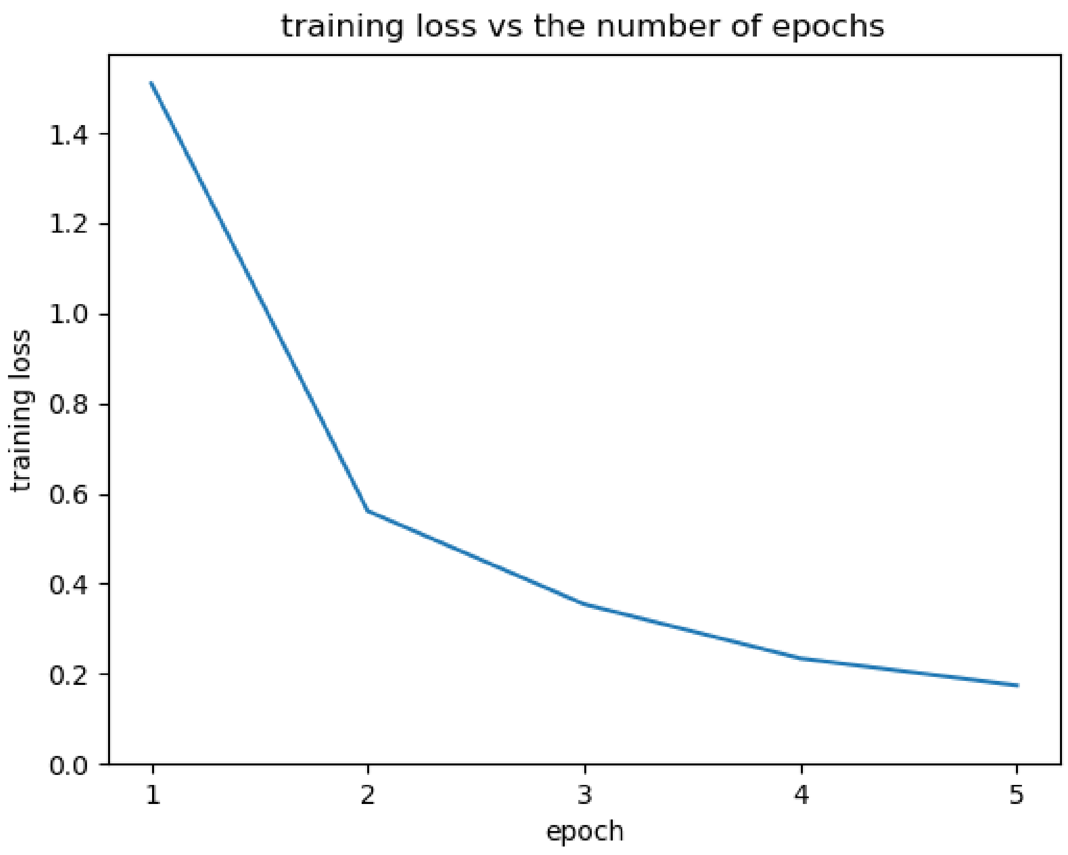
b.


training accuracy vs the number of epochs
training accuracy
epoch
1.0
0.8
0.6
0.4
0.2
0.0
1
2
3
4
5

c.

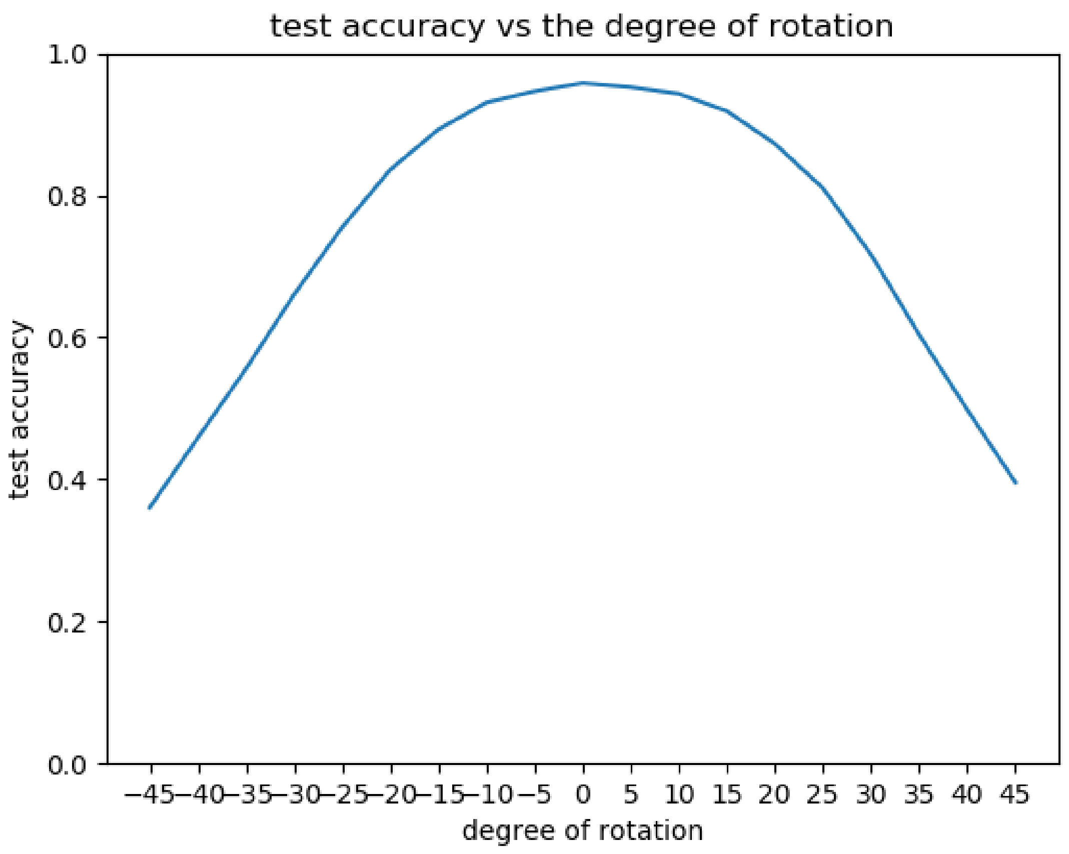

test loss vs the number of epochs
test loss
epoch
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
1
2
3
4
5

d.


training loss vs the number of epochs
training loss
epoch
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1
2
3
4
5

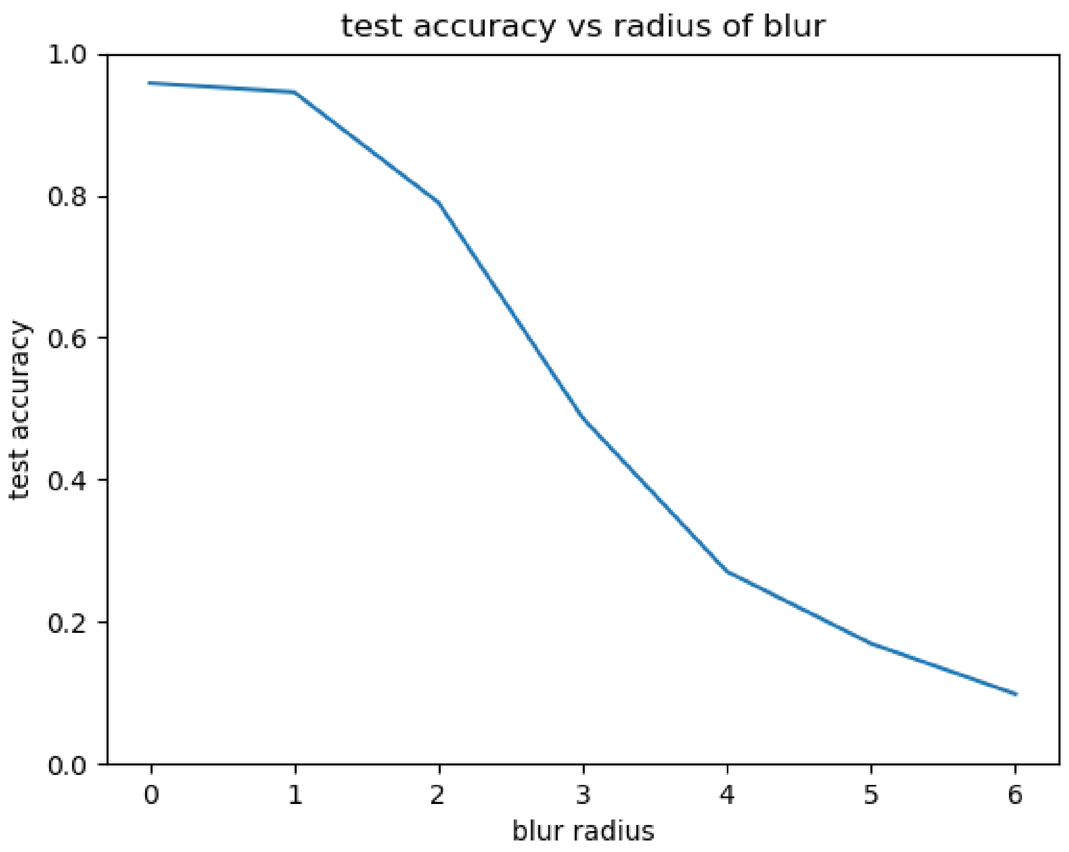
3. Code can be found in **VGG11_rotate_blur.py**. Ensure that you have **pillow** installed in addition to the packages in part 1. Instructions to run are the same. Google Colab instance:
   https://colab.research.google.com/drive/19zjWWgV6SrS-M03TSS0vn-X0Kf5ZeZ3T

   e.



As the images in our test set becomes more rotated in either direction, it looks less like any of the training samples we trained our CNN with, and consequently, our CNN becomes less and less able to correctly classify it, and the test accuracy decreases.

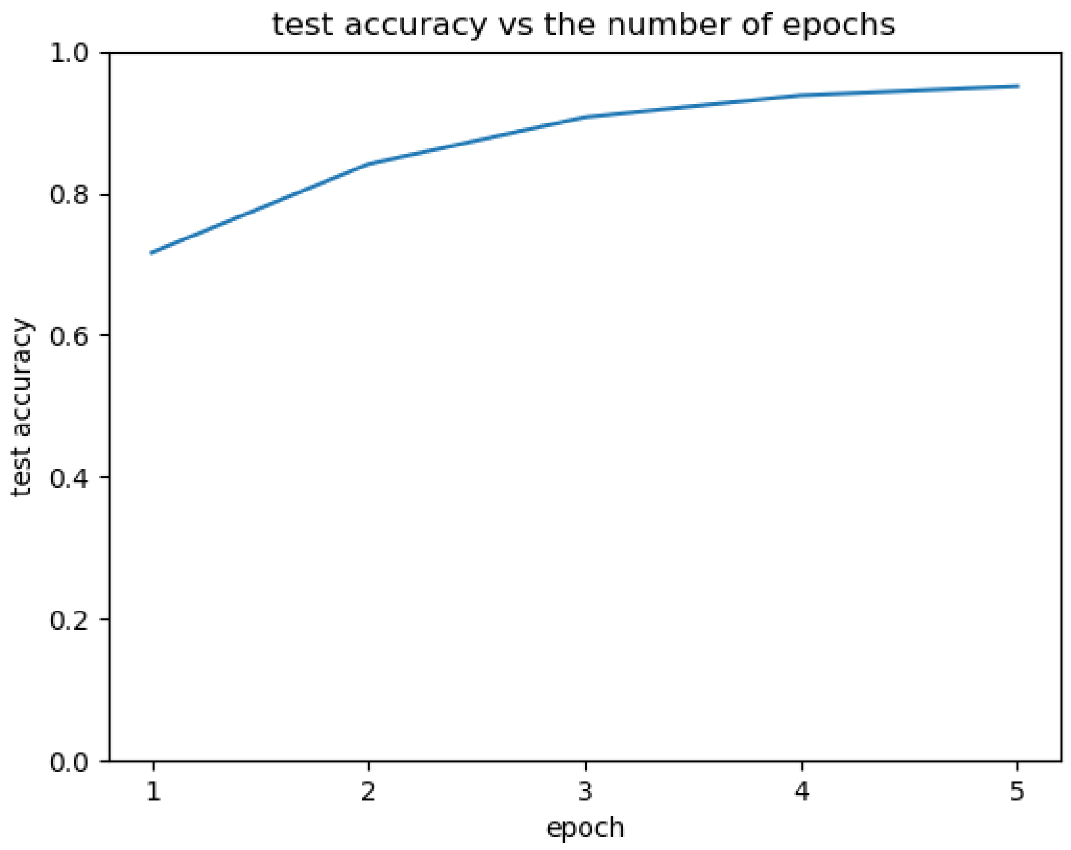
f.

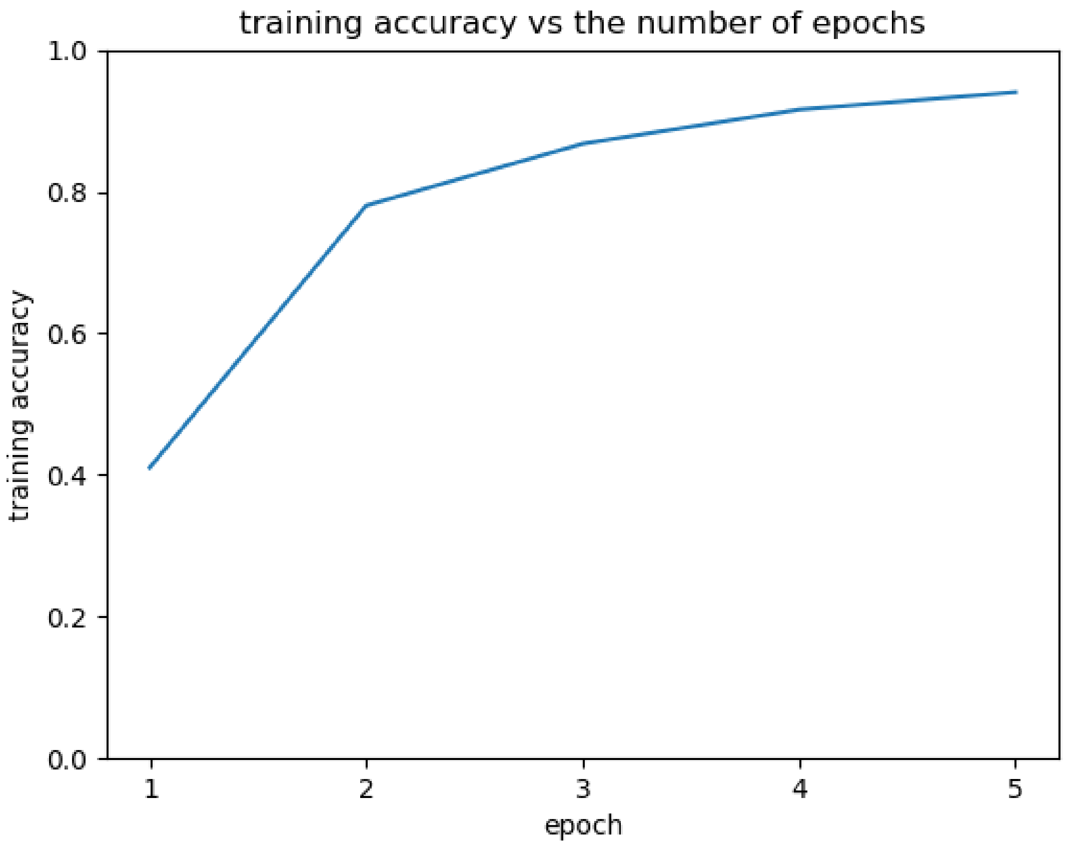


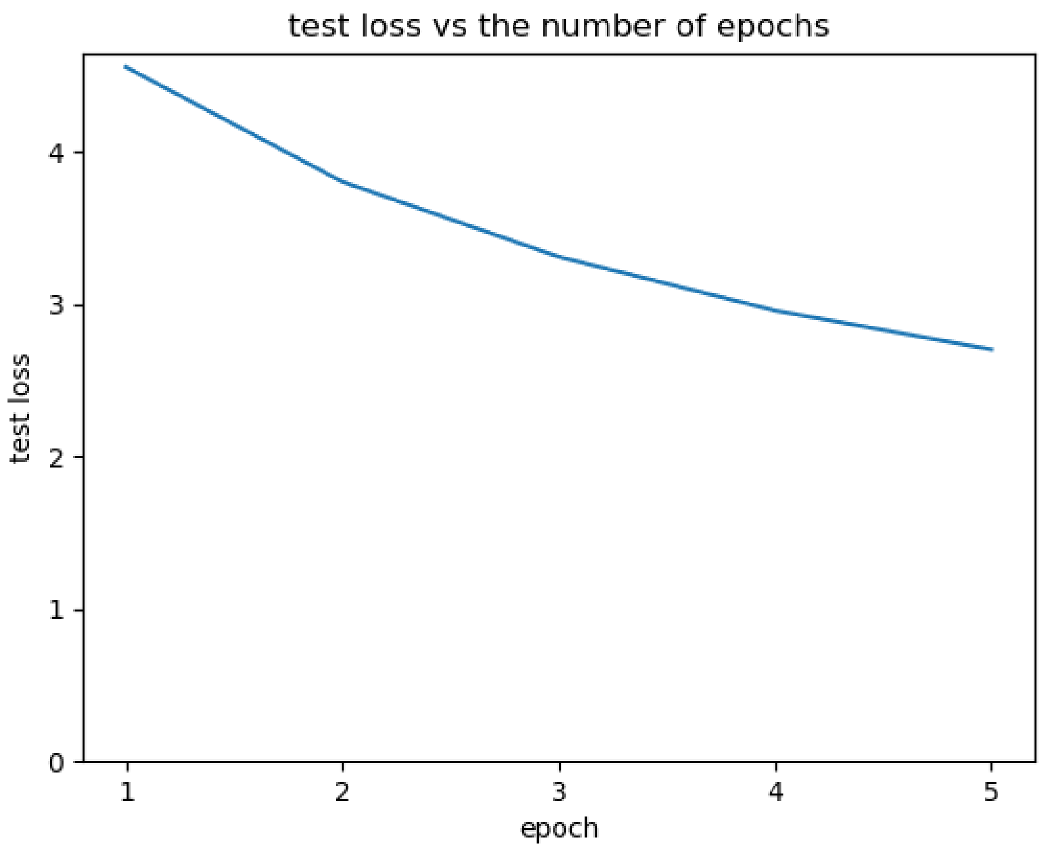
As the blur radius increases, not only do the images in our test set look less than the training samples used, but it is also harder to distinguish between different digits (e.g. 1 vs 7, 6 vs 8 vs 9) because of the blur. Consequently, our CNN becomes less and less able to correctly classify digits in the test set, and the test accuracy decreases.
The test accuracy also begins to taper off and converge to random guessing as the blur radius increases. This is because as the images become more blurred, they all begin to look indistinguishable from each other, (i.e. all digits just look like a blob of white), so the CNN can do no better than random guessing (0.1 accuracy).

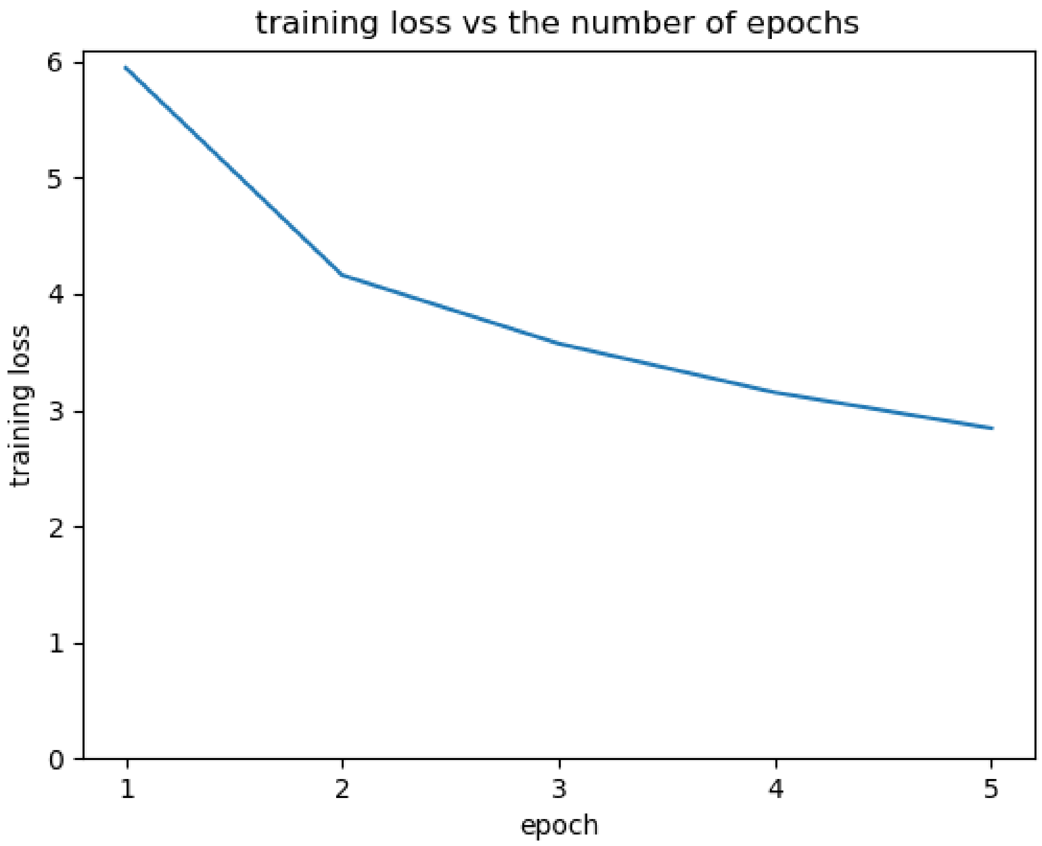
4.

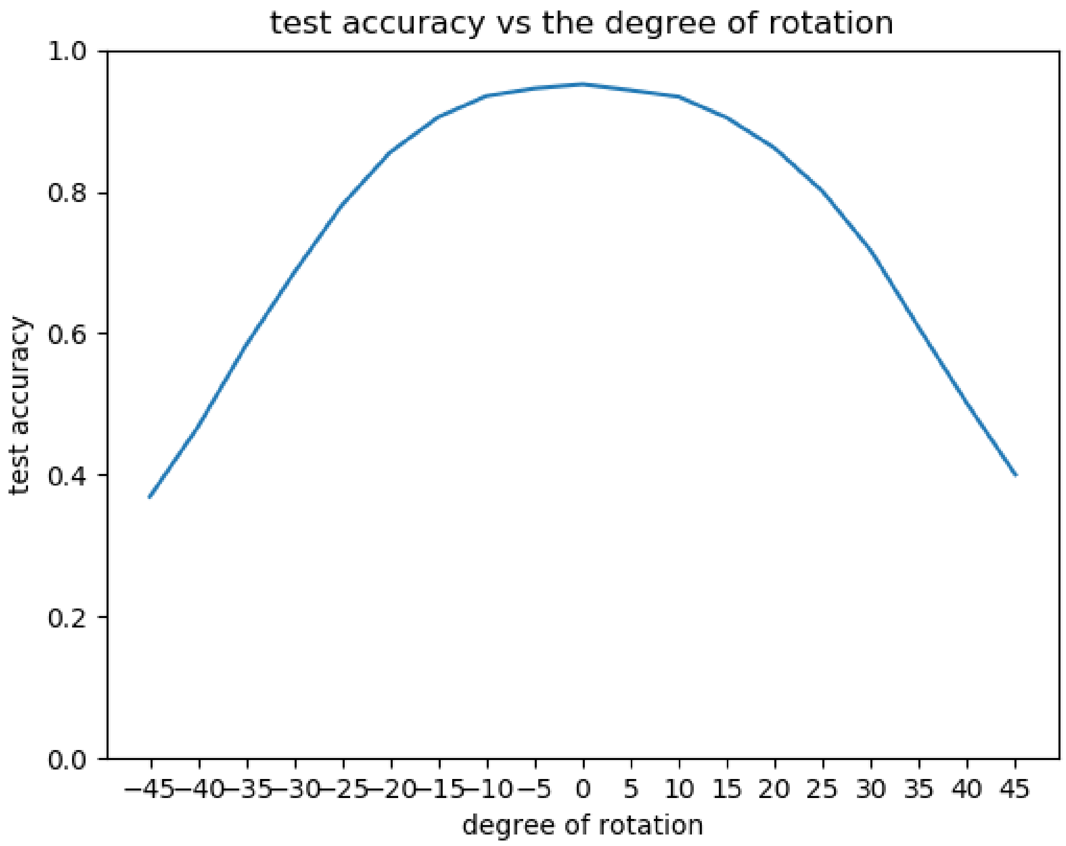
g. Code can be found in **VGG11_regularization.py.** Instructions to run are the same.
Google Colab instance:
https://colab.research.google.com/drive/1-WuTDEkxw05cL68bfYzEiJZFxWtUMhO4
Regularization constant used was **0.0005**, same as what was used in the original paper.
L2 kernel (weights) regularization was added to all convolutional AND fully connected layers.



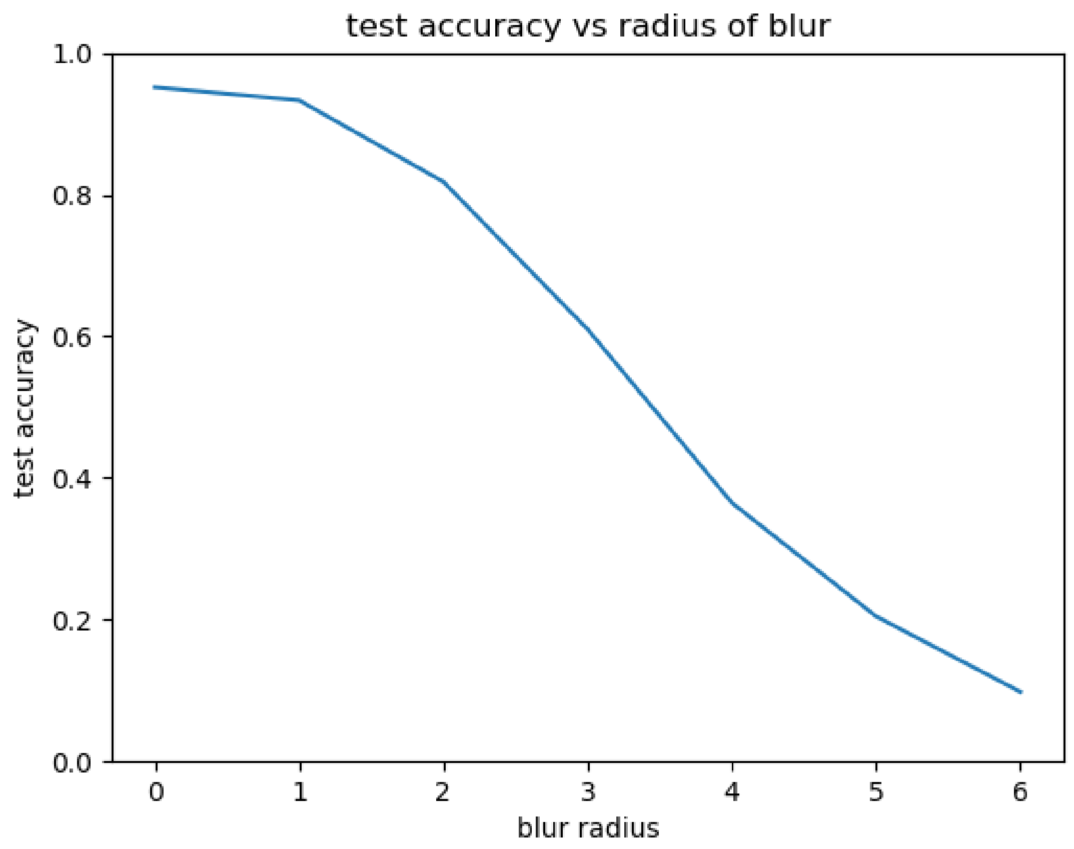

training accuracy vs the number of epochs
training accuracy
epoch
1.0
0.8
0.6
0.4
0.2
0.0
1
2
3
4
5


test loss vs the number of epochs
test loss
epoch
0
1
2
3
4
1
2
3
4
5


training loss vs the number of epochs
training loss
epoch
0
1
2
3
4
5
6
1
2
3
4
5

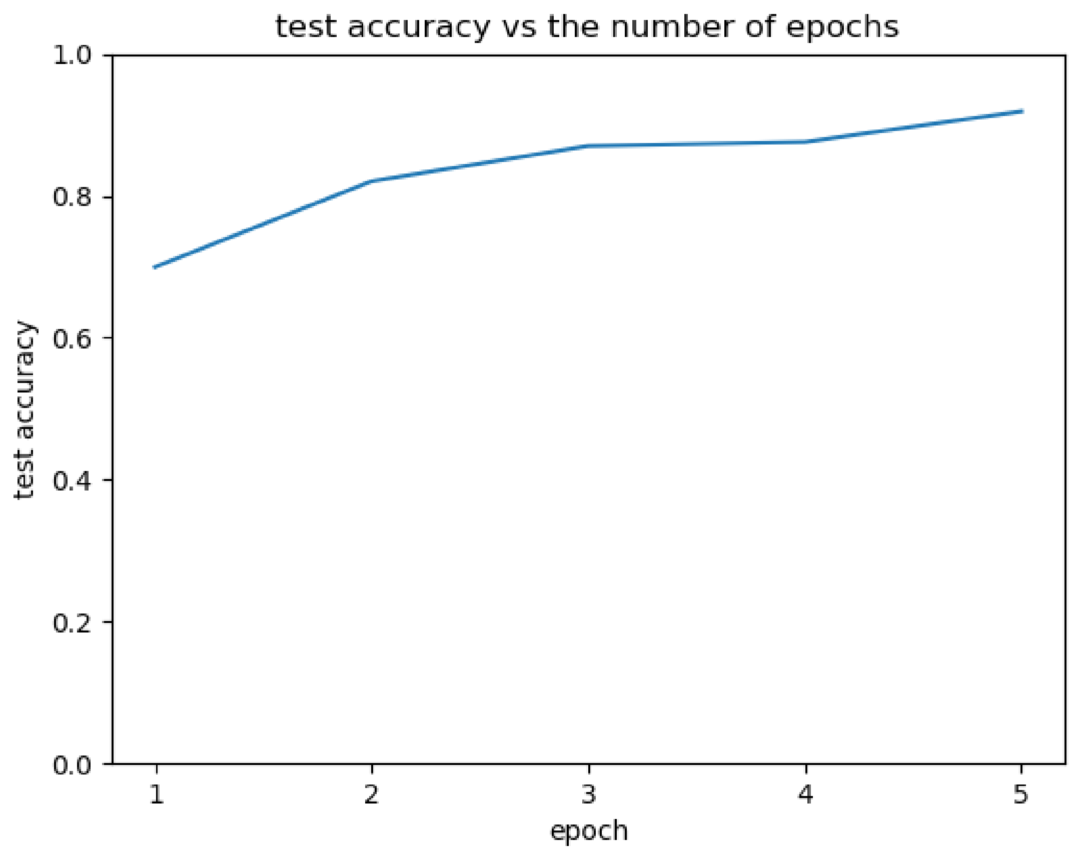


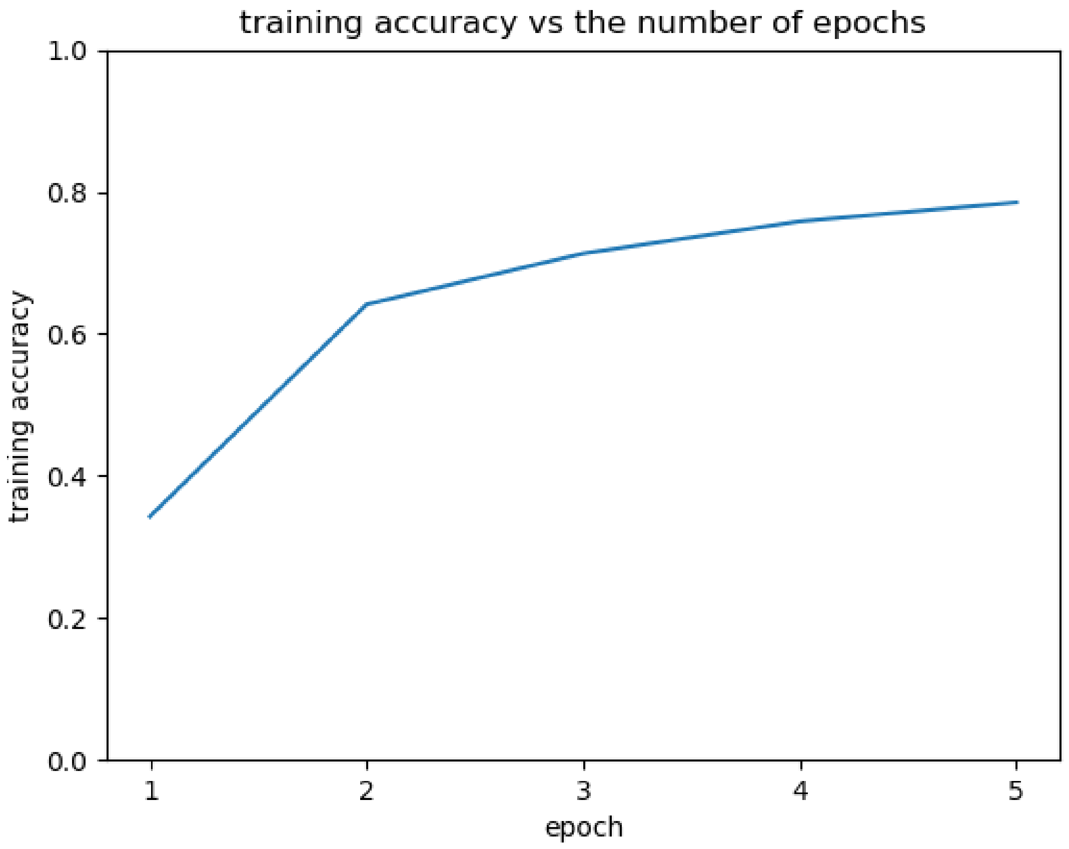
Regularization seemed to have no effect on the test accuracy vs degree of rotation. All values were very similar to the un-regularized graph.

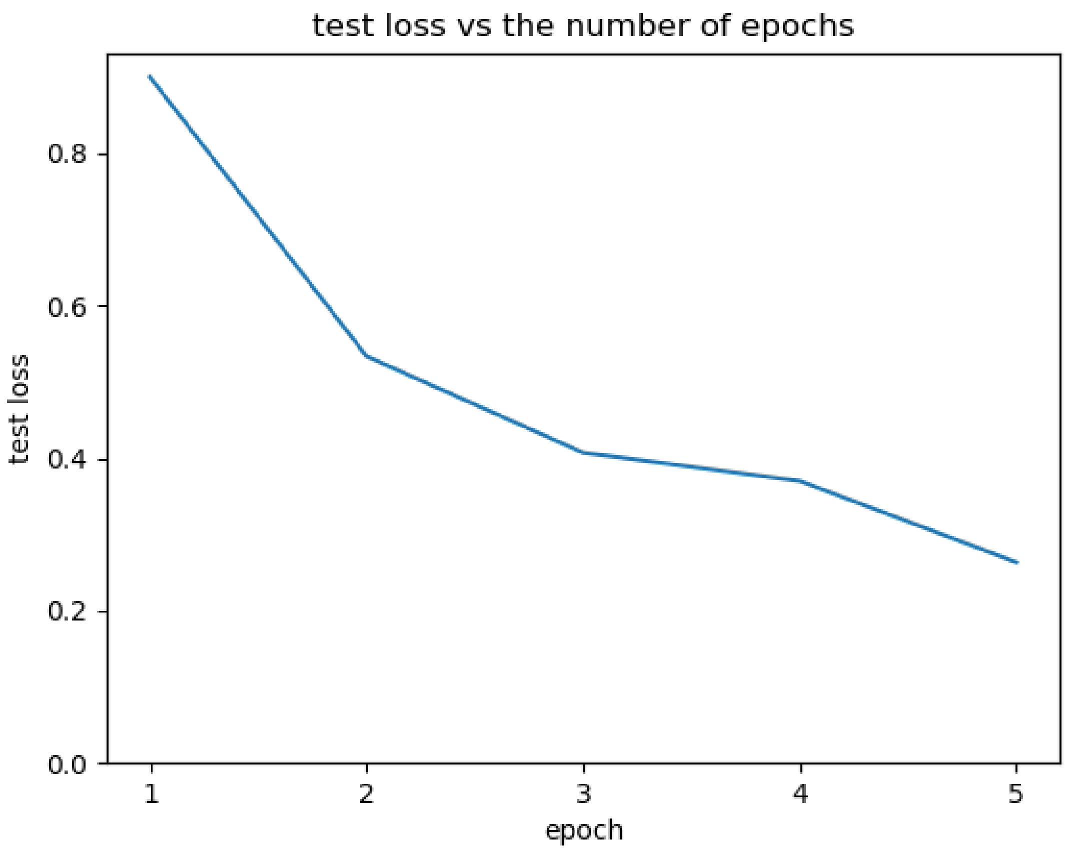


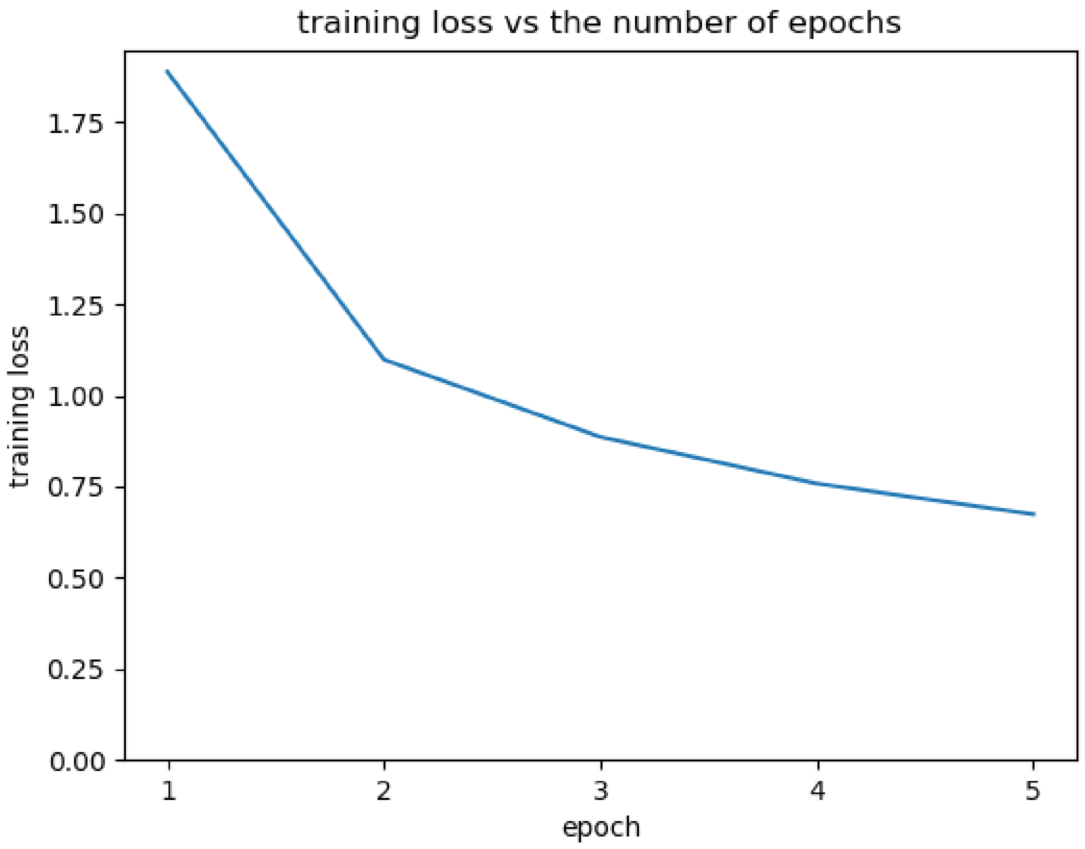
Similar to the test accuracy vs degree of rotation plot, regularization seemed to have no effect on the test accuracy with respect to the radius of the Gaussian blur applied to the test set.

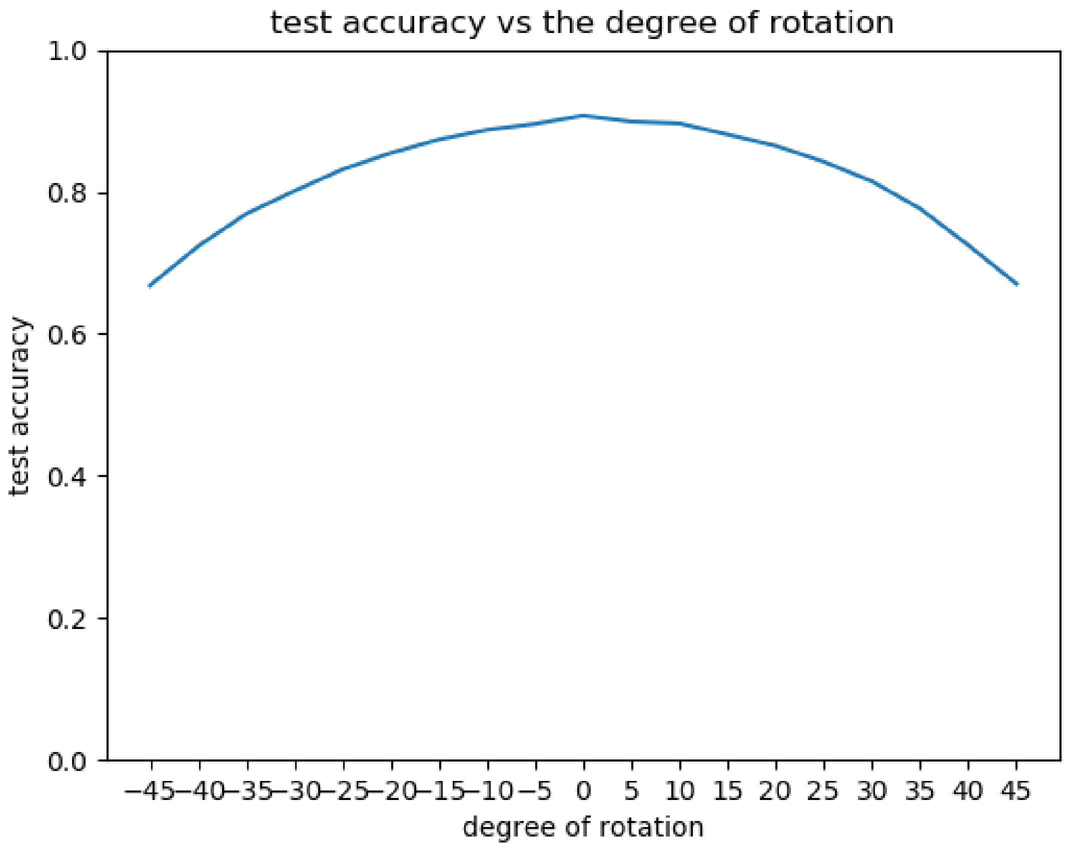
h. Code can be found in **VGG11_data_augmentation.py.** Instructions to run are the same. Google Colab instance:
https://colab.research.google.com/drive/1TQj2CtcKCTdME6kGxZ6Eef49m1zIxYjJ
Data augmentation was performed as such:
For each sample in the training set, a random number between 0 and 1 was generated.
If the number was greater than 0.5, a random degree of rotation from {-45,-40,-35,-30,-25,-20,-15,-10,-5,0,5,10,15,20,25,30,35,40,45} was chosen, and the image was rotated.
Similarly, for each sample in the training set, another random number was generated, and if greater than 0.5, a random radius of blur from {0, 1, 2, 3, 4, 5, 6} was chosen, and the image was blurred appropriately.
Note that these two processes were done independently, so samples could be unaffected, only rotated by some amount, only blurred by some amount, or blurred and rotated by some amounts.
This means that our resulting dataset has ~50% images that are not blurred or rotated, ~25% images that are only blurred or rotated, and ~25% that are blurred and rotated by some amount.



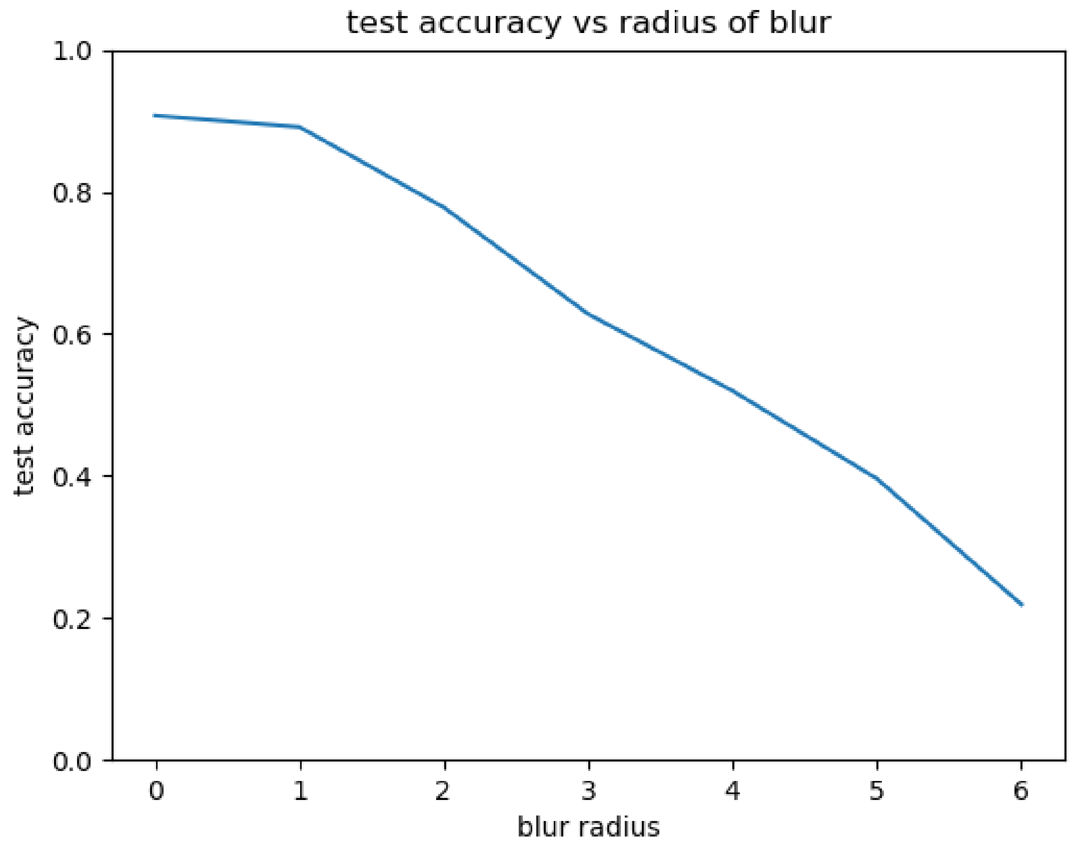

training accuracy vs the number of epochs
training accuracy
epoch
1.0
0.8
0.6
0.4
0.2
0.0
1
2
3
4
5


test loss vs the number of epochs
test loss
epoch
0.0
0.2
0.4
0.6
0.8
1
2
3
4
5


training loss vs the number of epochs
training loss
epoch
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
1
2
3
4
5



Using this augmentation method, our CNN becomes much better at classifying heavily rotated images (e.g. ~30% accuracy to ~60% accuracy for 45 degrees of rotation), at the slight expense of classifying un-rotated/lightly rotated images (e.g. ~95% accuracy to ~90% accuracy for un-rotated images – i.e. the unmodified dataset). The test accuracy vs degree of rotation curve is essentially "flattened".

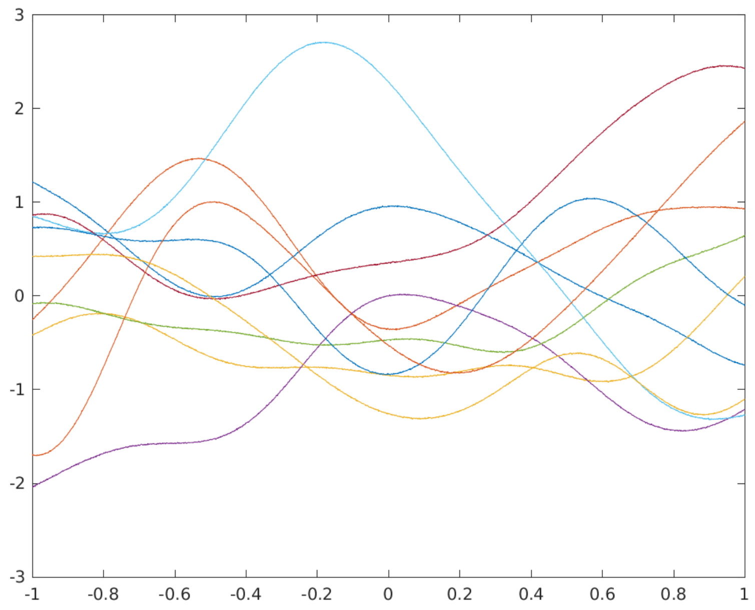


Using this augmentation method, test accuracy at smaller blurs radii (between 0-3) are unaffected. This may be because images with slight blurs are still distinguishable enough, and look close enough to their un-blurred counterparts such that a model trained on an un-augmented dataset can still perform well enough. However, the model makes some slight improvements on the more heavily blurred test sets (e.g. ~10% accuracy to ~20% on the dataset blurred with a radius of 6 – i.e. an improvement from random guessing). This may be because, although the heavily blurred digits can no longer be distinguishable from each other, they can still be separable into different classes (e.g. a blurred 6 and 8 may look alike, but neither will look like a blurred 1). These differences could have been picked up by the CNN, which allowed it to do better than random guessing.

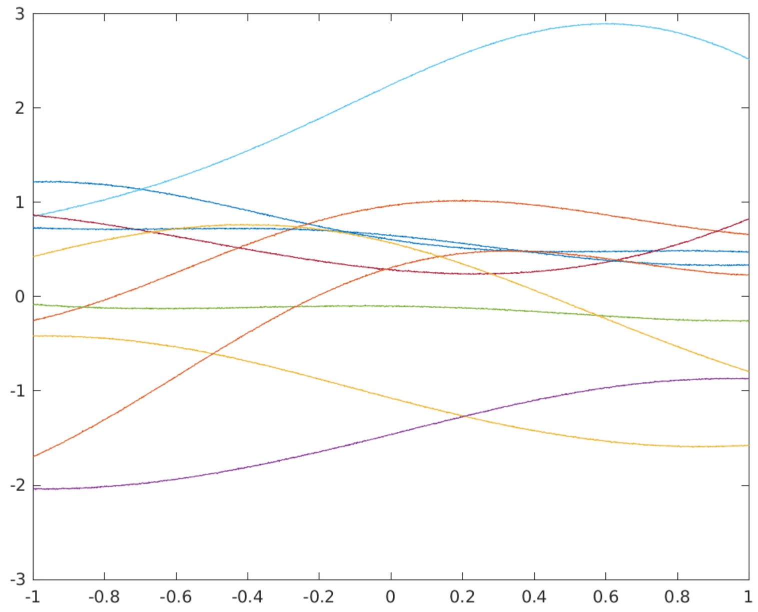
Exercise 2:

See test.m and gp.m for the matlab code to generate the graphs.

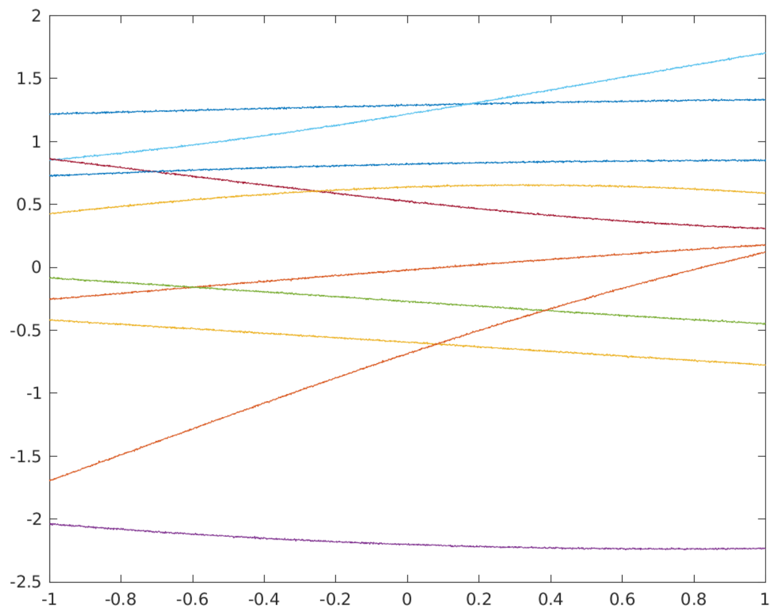
a) σ = 0.1:



σ = 1:



σ = 10:



As we increase σ, the sample paths get flatter and flatter – large curves and bumps are smoothed out.

b)

$$\frac{d^2}{ds\,dt} K_\sigma(s,t) = \frac{d}{ds}\left(\frac{d}{dt} e^{\frac{-(s-t)^2}{2\sigma}}\right)$$

$$\frac{d}{dt} e^{\frac{-(s-t)^2}{2\sigma}} = e^{\frac{-(s-t)^2}{2\sigma}} \cdot \frac{-2}{2\sigma}(s-t)\cdot(-1)$$ by chain rule.

$$= e^{\frac{-(s-t)^2}{2\sigma}} \cdot \frac{1}{\sigma}(s-t)$$

$\therefore$

$$\frac{d^2}{ds\,dt} K_\sigma(s,t) = \frac{d}{ds} e^{\frac{-(s-t)^2}{2\sigma}} \cdot \frac{1}{\sigma}(s-t)$$

$$= \frac{1}{\sigma}(s-t)\frac{d}{ds} e^{\frac{-(s-t)^2}{2\sigma}} + e^{\frac{-(s-t)^2}{2\sigma}} \frac{d}{ds}\frac{1}{\sigma}(s-t)$$

by product rule

$$= \frac{1}{\sigma}(s-t) e^{\frac{-(s-t)^2}{2\sigma}} \cdot \frac{-1}{\sigma}(s-t)\cdot(1) +$$ by chain rule

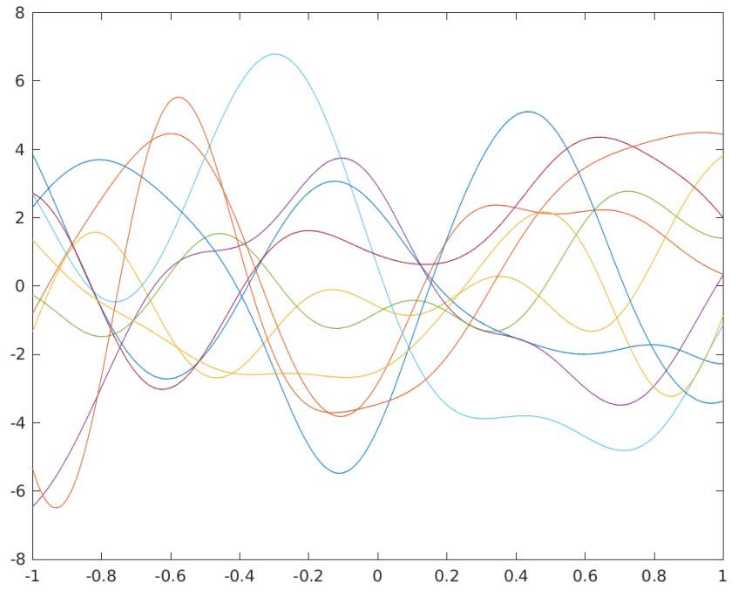
$$e^{\frac{-(s-t)^2}{2\sigma}} \cdot \frac{1}{\sigma}$$

$$= -\frac{(s-t)^2}{\sigma^2} e^{\frac{-(s-t)^2}{2\sigma}} + e^{\frac{-(s-t)^2}{2\sigma}} \cdot \frac{1}{\sigma}$$

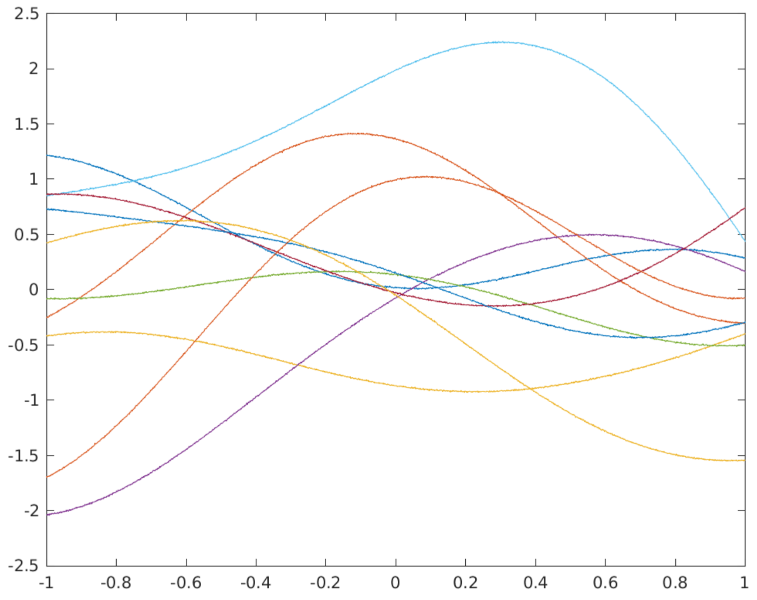
$$= e^{\frac{-(s-t)^2}{2\sigma}}\left(\frac{1}{\sigma} - \frac{(s-t)^2}{\sigma^2}\right)$$ factor out $e^{\frac{-(s-t)^2}{2\sigma}}$

$$\boxed{= e^{\frac{-(s-t)^2}{2\sigma}} \cdot \frac{1}{\sigma}\left(1 - \frac{(s-t)^2}{\sigma}\right)}$$ factor out $\frac{1}{\sigma}$

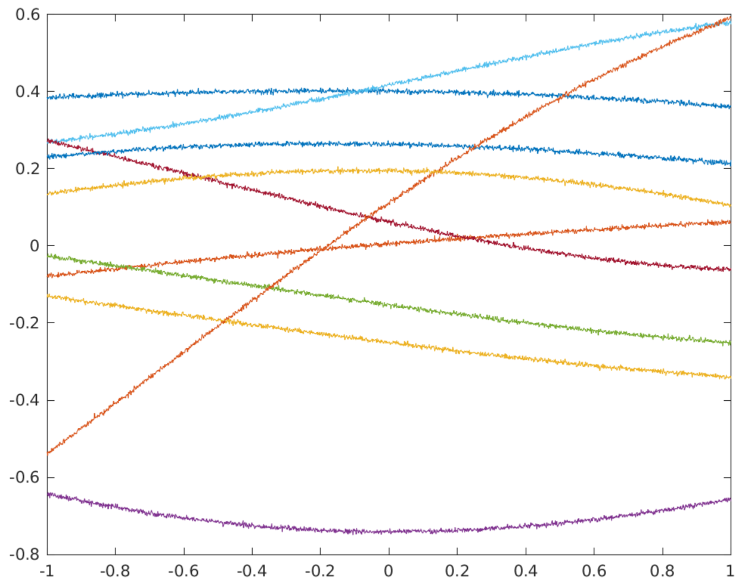
σ = 0.1:


8
6
4
2
0
-2
-4
-6
-8
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

σ = 1:


2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

σ = 10:



Similar to part a), as we increase σ, the sample paths get flatter and flatter – "stretched out" in some sense. Under the mixed partial derivative covariance, the scale of the amplitudes of the curves also decrease – notice how the range of the y axis go from [-8,8] to [-0.8, 0.6] as σ increases.

Comparing the sample paths in part (a) and (b), the sample paths look very similar for each σ – the paths have similar curves at similar spots along the x axis. This is expected, as looking closely, we can see that κ'_σ is basically κ_σ, scaled by a function of σ (technically a function of 1/σ), so the curves will look very similar.

where $\tilde{x}_i = x_i + \epsilon_i$,
$\epsilon_i \sim N(0, \lambda \mathbb{I})$

3(a) Simplifying $\min_{w \in \mathbb{R}^d} \sum_{i=1}^{n} E[(y_i - w^T \tilde{x}_i)^2]$,

$= \min \sum E[y_i^2 - 2y_i w^T \tilde{x}_i + (w^T \tilde{x}_i)^2]$ (expand square)

$= \min \sum E[y_i^2 - 2y_i w^T (x_i + \epsilon_i) + (w^T(x_i + \epsilon_i))^2]$ (defn of $\tilde{x}_i$)

$= \min \sum E[y_i^2 - 2y_i w^T x_i - 2y_i w^T \epsilon_i + (w^T x_i + w^T \epsilon_i)^2]$ (linearity of dot product))

$= \min \sum E[y_i^2 - 2y_i w^T x_i - 2y_i w^T \epsilon_i + (w^T x_i)^2 + 2w^T x_i w^T \epsilon_i + (w^T \epsilon_i)^2]$
(expand square)

$= \min \sum E[y_i^2 - 2y_i w^T x_i + (w^T x_i)^2 - 2y_i w^T \epsilon_i + 2w^T x_i w^T \epsilon_i + (w^T \epsilon_i)^2]$
(rearrange terms)

$= \min \sum E[(y_i - w^T x_i)^2 - 2y_i w^T \epsilon_i + 2w^T x_i w^T \epsilon_i + (w^T \epsilon_i)^2]$
(simplify square)

$= \min \sum E[(y_i - w^T x_i)^2] - \sum E[2y_i w^T \epsilon_i - 2w^T x_i w^T \epsilon_i - (w^T \epsilon_i)^2]$
(linearity of Σ, $E[x]$)

$= \min \underbrace{\sum (y_i - w^T x_i)^2}_{} - \sum E[2y_i w^T \epsilon_i] - E[2w^T x_i w^T \epsilon_i] - E[(w^T \epsilon_i)^2]$
(linearity of $E[x]$)

since y_i, w, x_i are not random variables, so $E[(y_i - w^T x_i)^2] = (y_i - w^T x_i)^2$

Since $\epsilon_i \sim N(0, \lambda \mathbb{I})$, $E(\epsilon_i) = 0$, $E(\epsilon_i^2) = \lambda \mathbb{I}$

Also note that $(w^T \epsilon_i)^2 = \sum_{j=1}^{d} (w_j)^2 (\epsilon_{ij})^2 + \sum_{j=1}^{d} \sum_{k=j+1}^{d} 2w_j w_k \epsilon_{ij} \epsilon_{i,k}$

Then, our second term $\sum E[2y_i w^T \epsilon_i] - E[2w^T x_i w^T \epsilon_i] - E[(w^T \epsilon_i)^2]$ simplifies to:

$\sum 2y_i w^T E[\epsilon_i] - 2w^T x_i w^T E[\epsilon_i] - \sum w_j^2 E[(\epsilon_{ij})^2] - \sum\sum 2w_j w_k E[\epsilon_{ij}] E[\epsilon_{ik}]$

$= \sum 0 - 0 - \lambda \|w\|_2^2 - 0$

$= \sum_i^n -\lambda \|w\|_2^2 = -\lambda n \|w\|_2^2$.

$\therefore$ our expression simplifies to

$\min_{w \in \mathbb{R}^d} \sum_i^n (y_i - w^T x_i)^2 + \lambda n \|w\|_2^2$

(b) Simplifying $\min_{w \in \mathbb{R}^d} \sum_{i=1}^{n} E[(y_i - w^T \tilde{x}_i)^2]$, where $\tilde{x}_i = x_i \odot \epsilon_i$, $\epsilon_{ij} \sim \text{Bernoulli}(p)/p$

$= \min \sum E[\, y_i^2 - 2y_i w^T \tilde{x}_i + (w^T \tilde{x}_i)^2]$ (expand squares)

$= \min \sum E[y_i^2] - E[2y_i w^T \tilde{x}_i] + E[(w^T \tilde{x}_i)^2]$ (linearity of $E[x]$)

① $= \min \sum y_i^2 - 2y_i E[w^T \tilde{x}_i] + E[(w^T \tilde{x}_i)^2]$ (expectation of constants)

Consider $E[w^T \tilde{x}_i] = E[\sum_{j=1}^{d} w_j \tilde{x}_{i,j}]$ (definition of dot product)

$= E[\sum w_j x_{ij} \epsilon_{ij}]$ (definition of $\tilde{x}_i$)

$= \sum w_j x_{ij} E[\epsilon_{ij}]$ (expectation of constants + linearity of $E[x]$)

Note that if $X \sim \text{Bernoulli}(p)$, $E[X] = p$.
Since $\epsilon_{ij} \sim \text{Bernoulli}(p)/p$, $E[\epsilon_{ij}] = p/p = 1$.
$\therefore$ we have $E[w^T \tilde{x}_i] = \sum w_j x_{ij} \cdot 1$ (defn of $E[\epsilon_{ij}]$)

$= \sum w_j x_{ij}$

$= w^T x_i$ ② (defn of dot product)

Consider $E[(w^T \tilde{x}_i)^2] = E[\sum_{j=1}^{d} w_j^2 \tilde{x}_j^2 + \sum_{j=1}^{d} \sum_{k=j+1}^{d} 2 w_j w_k \tilde{x}_j \tilde{x}_{ik}]$ (defn of $(w^T x)^2$, from part (a))

$= \sum w_j^2 x_{ij}^2 E[\epsilon_{ij}^2] + 2 \sum \sum w_j w_k x_{ij} x_{ik} \underbrace{E[\epsilon_{ij}] E[\epsilon_{ik}]}$

(defn of $\tilde{x}_i$ + linearity of $E[x]$ + expectation of constants) since ϵ_{ij}'s are drawn independently

By definition, $E[X^2] = \sum_{x \in S} x^2 p(x)$, for discrete R.Vs X, so

$E[\epsilon^2] = \sum_{\epsilon \in \{0, 1/p\}} \epsilon \, p(\epsilon)$

$= 0^2(1-p) + (1/p)^2(p) = \cancel{p}/p^2 = 1/p$

So $E[(w^T \tilde{x}_i)^2]$ simplifies to

$$\frac{1}{p} \sum_{j=1}^{d} w_j^2 x_{ij}^2 + 2 \sum_{j=1}^{d} \sum_{k=j+1}^{d} w_j x_{ij} .$$

Now consider $$(w^Tx_i)^2 - E[(w^T\tilde{x}_i)^2] = \overbrace{\sum_{j=1}^{d} w_j^2 x_{ij}^2 + 2\sum_{j=1}^{d}\sum_{k=j+1}^{d} w_j w_k x_{ij} x_{ik}}^{\text{defn of } (w^Tx)^2\text{, from part (a)}}$$
$$- \left(\frac{1}{p}\sum_{j=1}^{d} w_j^2 x_{ij}^2 + 2\sum_{j=1}^{d}\sum_{k=j+1}^{d} w_j w_k x_{ij} x_{ik}\right)$$
$$= \sum w_j^2 x_{ij}^2 - \frac{1}{p}\sum w_j^2 x_{ij}^2$$
$$= (1-\frac{1}{p})\sum w_j^2 x_{ij}^2$$

Rearranging gives:
$$E[(w^T\tilde{x}_i)^2] = (w^Tx_i)^2 - (1-\frac{1}{p})\sum w_j^2 x_{ij}^2 \quad ③$$

Plugging ②, ③ into ① gives:
$$\min_{w\in\mathbb{R}^d}\sum_{i=1}^{n} E[(y_i - w^T\tilde{x}_i)^2] = \min_{w\in\mathbb{R}^d}\sum_{i=1}^{n} y_i^2 - \underbrace{2y_iw^Tx_i}_{②} + \underbrace{(w^Tx_i)^2 - (1-\frac{1}{p})\sum w_j^2 x_{ij}^2}_{③}$$

$$= \min_{w\in\mathbb{R}^d}\left(\sum_{i=1}^{n} y_i^2 - 2y_iw^Tx_i + (w^Tx_i)^2\right) + \left(\sum_{i=1}^{n}(\frac{1}{p}-1)\sum_{j=1}^{d} w_j^2 x_{ij}^2\right)$$ (linearity of summations)

$$= \min_{w\in\mathbb{R}} \underbrace{\sum_{i=1}^{n}(y_i - w^Tx_i)^2}_{\text{least squares regression}} + \underbrace{(\frac{1}{p}-1)\sum_{i=1}^{n}\sum_{j=1}^{d} w_j^2 x_{ij}^2}_{\text{regularization constant}}$$ (simplify square + pull constant out of summation)