ML Final Project

Nico and Sarah

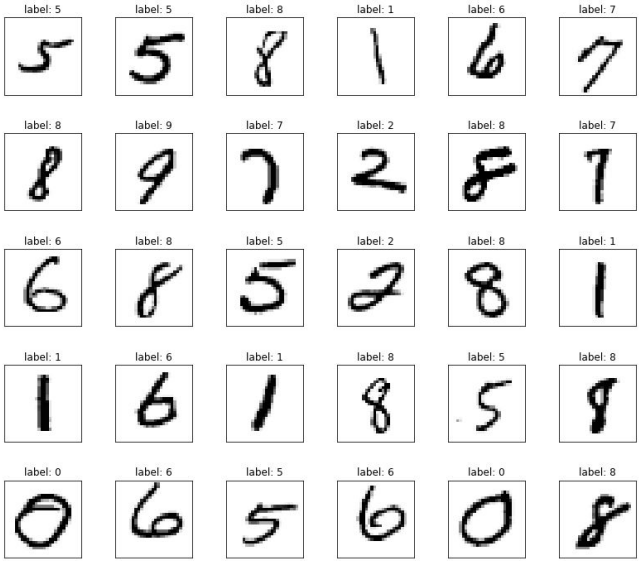
Why

Which classification model is best suited for image classification given different types of noise added on top of the image?

We wanted to test the efficacy of different classification models on altered image classification using the MNIST dataset. This research can be useful & notable in today's world as

Dataset

- MNIST Dataset
  - Handwritten numbers dataset
  - Widely used for image classification tasks
- Since it is widely used in image classification should work in our project (Validating the dataset)
- 70,000 images,
- Image comprised of 28 x 28 field, 784 pixel (features)
- Type - Float



Methods

We created a class for each type of the noise. We also did this so it could be integrated into scikit-learn pipelines.

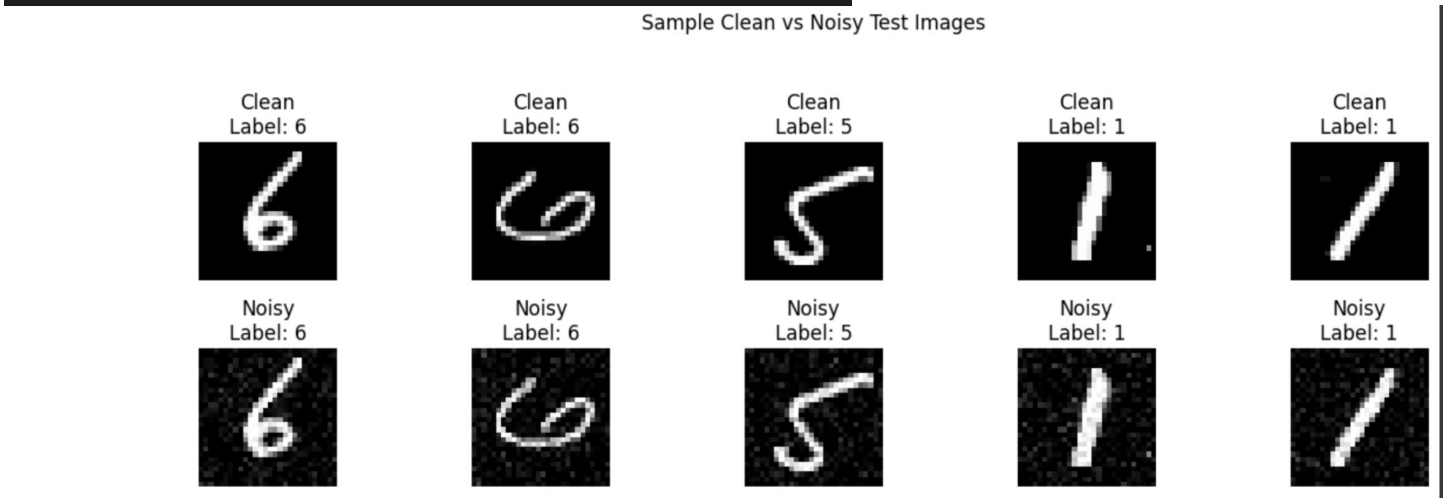
For each type of noise, we tested three different things. 1. The control, so training on clean, testing on clean. 2. We tested - Train on noisy, test on clean, 3. Train on noisy, test on noisy, 4. Train on clean, test on noisy. We did this by having two different pipelines - one that trains on noisy data and another that trains on clean. We pass both through our training function, and there they fit, predict, and print results

Implementation - Gaussian Noise (White noise)

```python
class AddGaussianNoise(BaseEstimator, TransformerMixin):
    # Controls the std of the noise distribution
    def __init__(self, noise_std=25):
        # In this case, about 68% of noise values will fall between -25 and 25
        self.noise_std = noise_std

    # Just a placeholder to comply with sklearn,
    # since most transformers need to learn parameters
    def fit(self, X, y=None):
        return self

    def transform(self, X):
        # Creates matrix of random values from Gaus dist,
        # Each image gets a unique random noise added to it
        noisy = X + np.random.normal(0, self.noise_std, X.shape)
        # Ensures it stays within the given pixel range for 8b image
        return np.clip(noisy, 0, 255)
```

Practical Applications

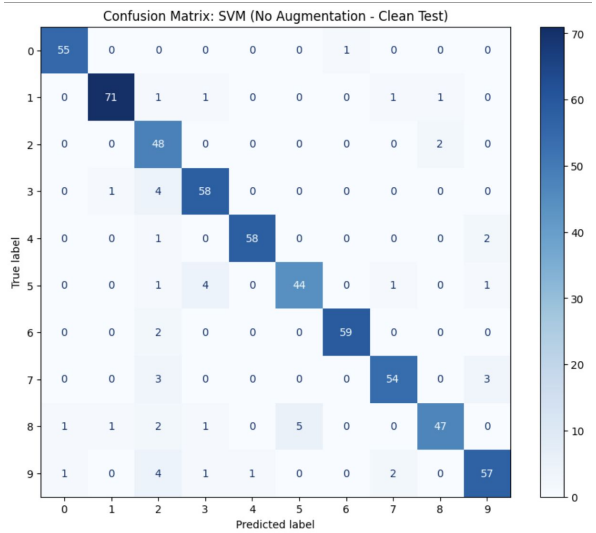
- Tests resilience to sensor or thermal noise
- MRI
- CT scans
- PET scans



Control Group - Train on clean, test on clean

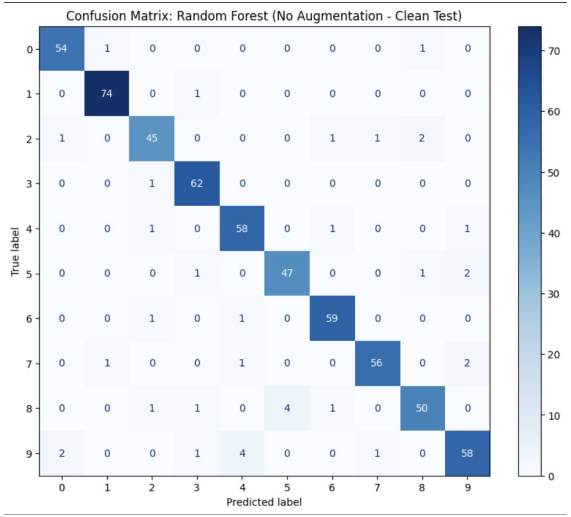
SVM

0.92



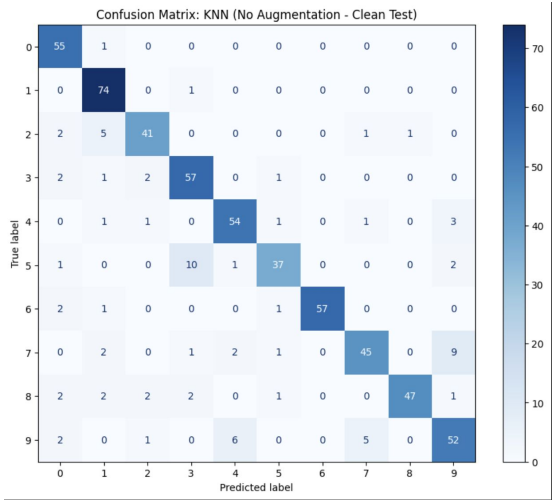
RF

0.94



KNN

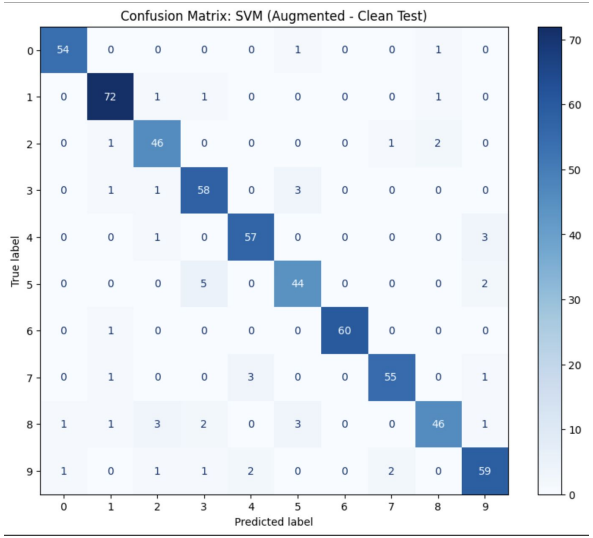
0.86



Results- Train on noisy, test on clean

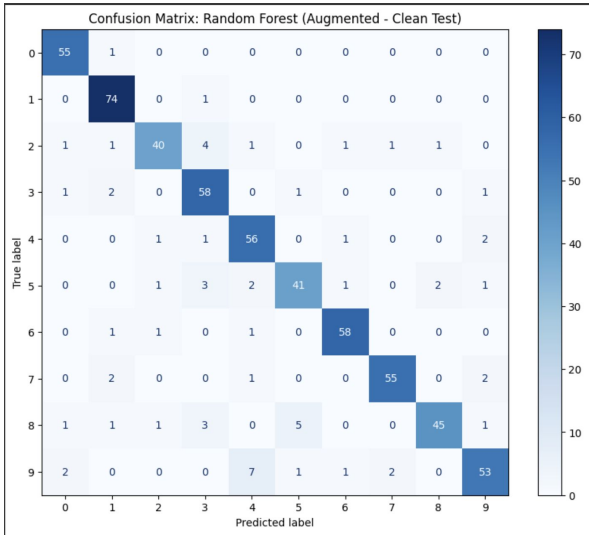
SVM

0.92



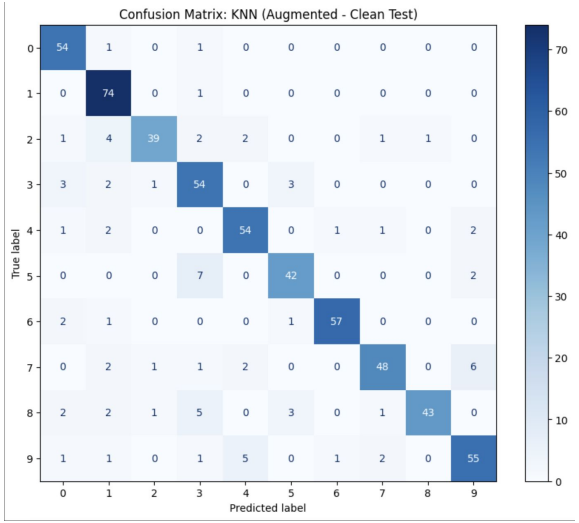
RF

0.89



KNN

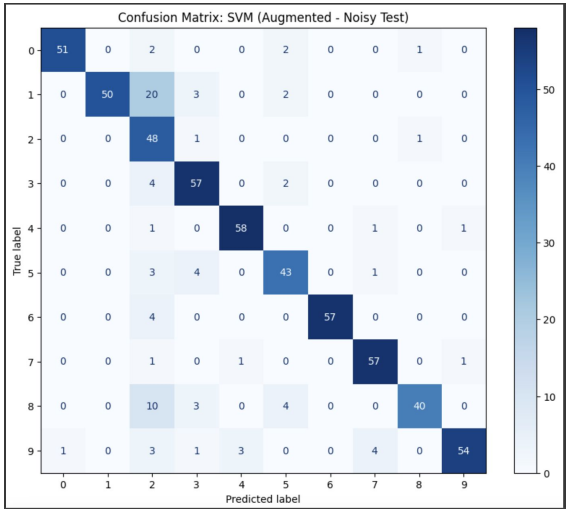
0.87



Results- Train on noisy, test on noisy

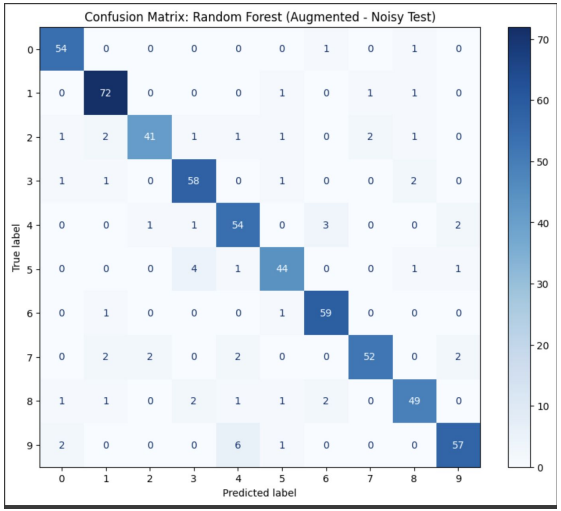
SVM

0.86



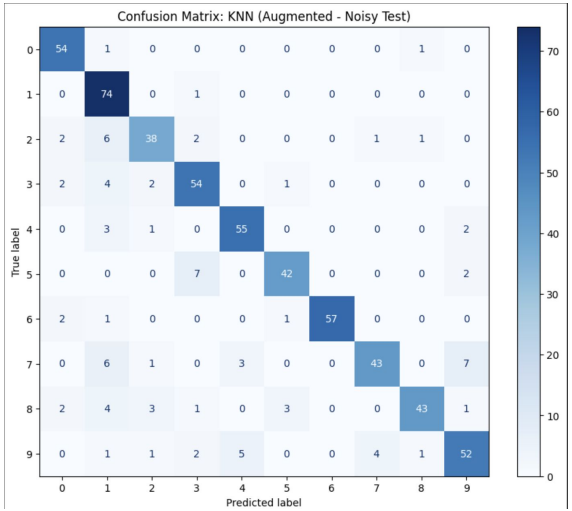
RF

0.90



KNN

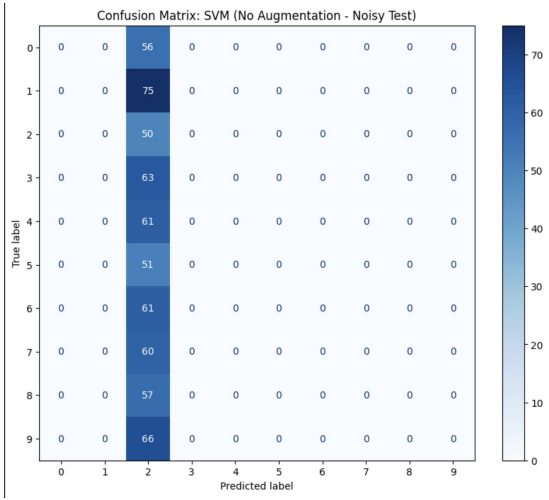
0.85



Results- Train on clean, test on noisy

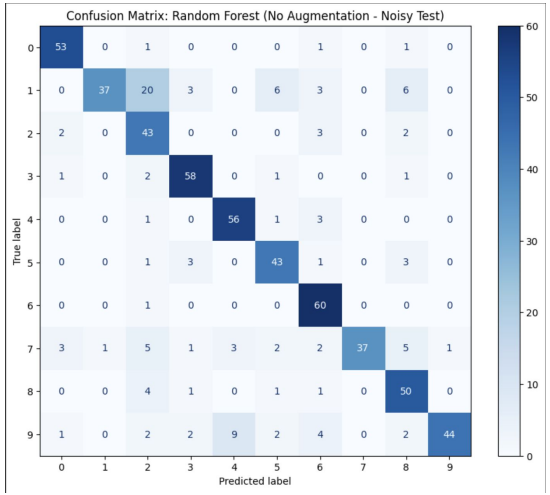
SVM

0.08



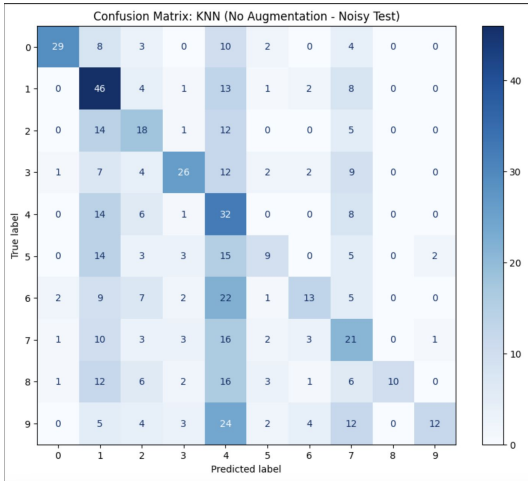
RF

0.80



KNN

0.36

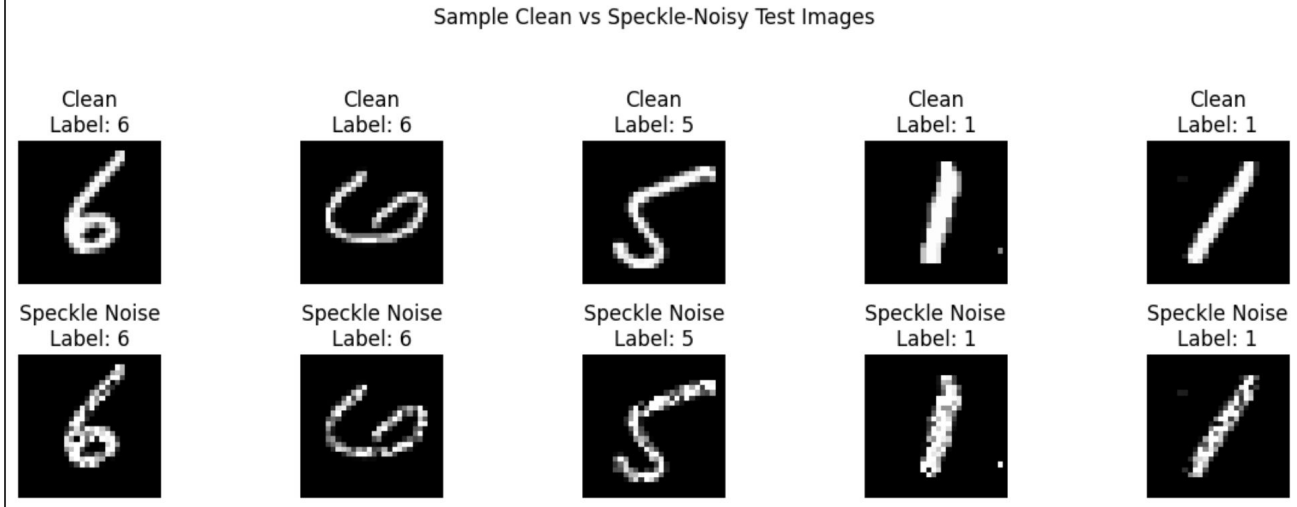


Speckle Noise

```python
class AddSpeckleNoise(BaseEstimator, TransformerMixin):
    # Initializes the transformer with a noise level
    def __init__(self, noise_level=0.5):
        self.noise_level = noise_level  # Controls intensity of speckle noise

    # Just a placeholder to comply with sklearn,
    # since most transformers need to learn parameters
    def fit(self, X, y=None):
        return self

    # Creates a noise matrix with same shape as X (i.e. images, pixels),
    # Fills it with random vals from a standard norm dist (i.e. N(0,1))
    def transform(self, X):
        # Multiplies the matrix by noise_level to control intensity
        noise = np.random.randn(*X.shape) * self.noise_level
        # The noise scales with the original pixel intensity (X)
        noisy = X + X * noise  # Speckle formula: I = I0 + I0 * noise
        # Ensures it stays within the given pixel range for 8b image
        return np.clip(noisy, 0, 255)
```

Practical Applications

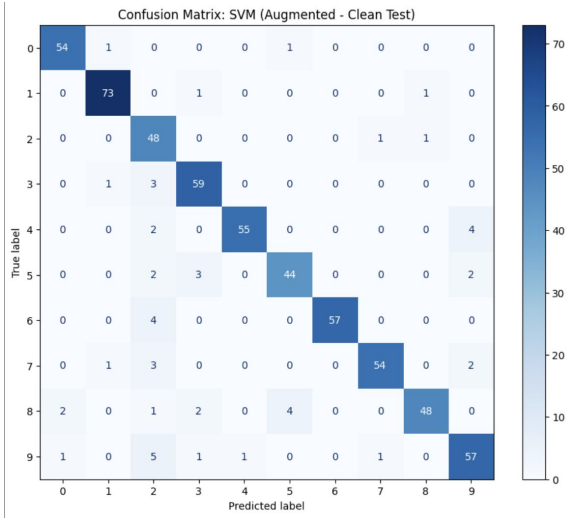
- Common in
- Radar Imaging
- Ultrasounds



Results - Train on noisy, test on clean

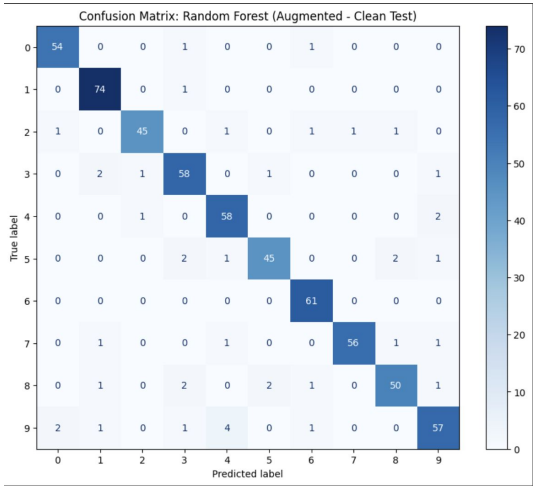
SVM

0.92



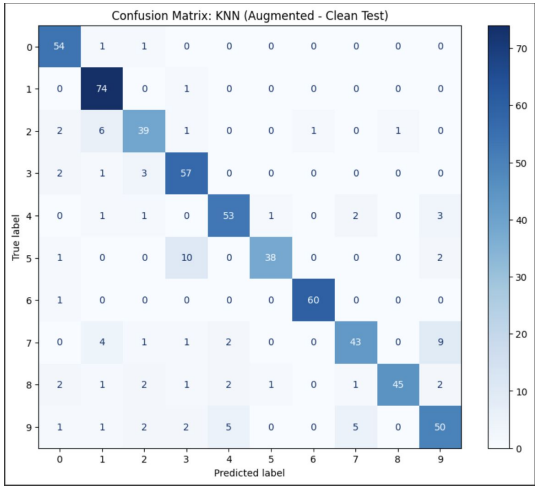
RF

0.93



KNN

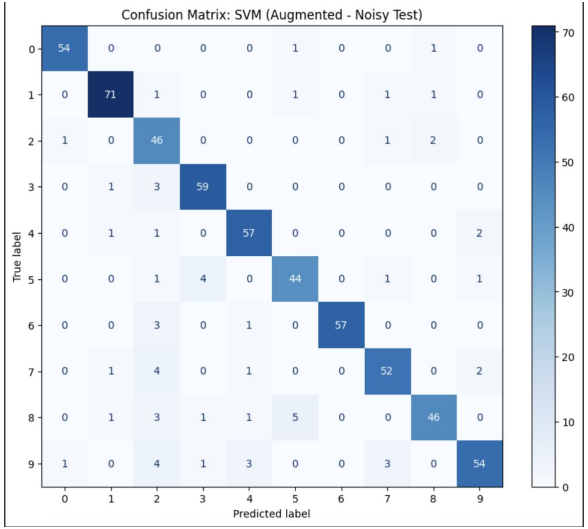
0.85



Results - Train on noisy, test on noisy

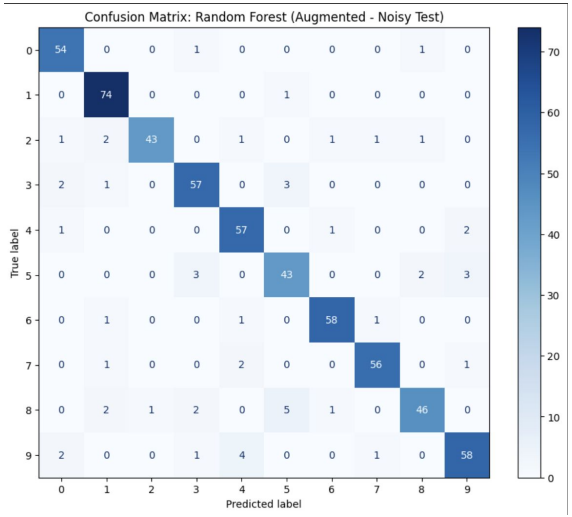
SVM

0.90



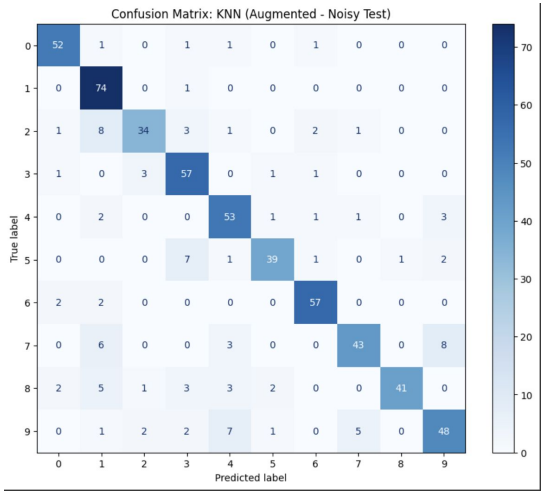
RF

0.91



KNN

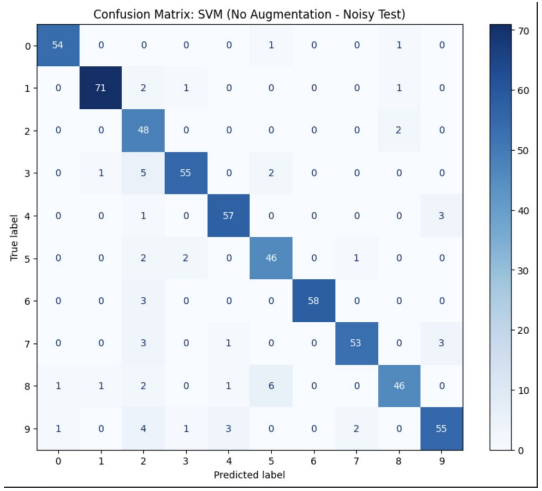
0.83



Results - Train on clean, test on noisy

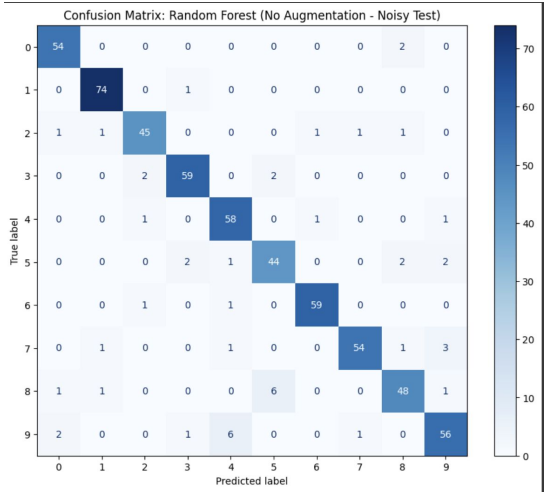
SVM

0.91



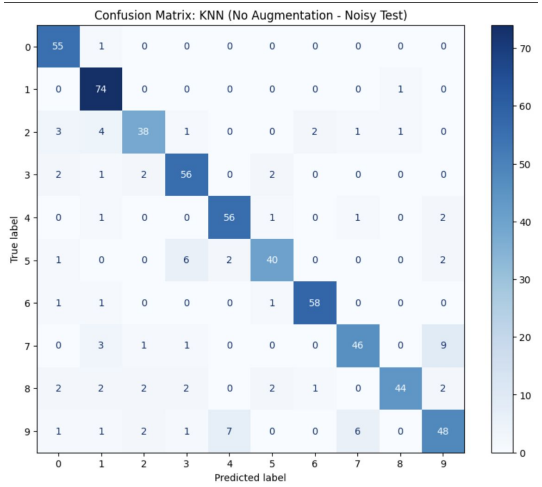
RF

0.92



KNN

0.86

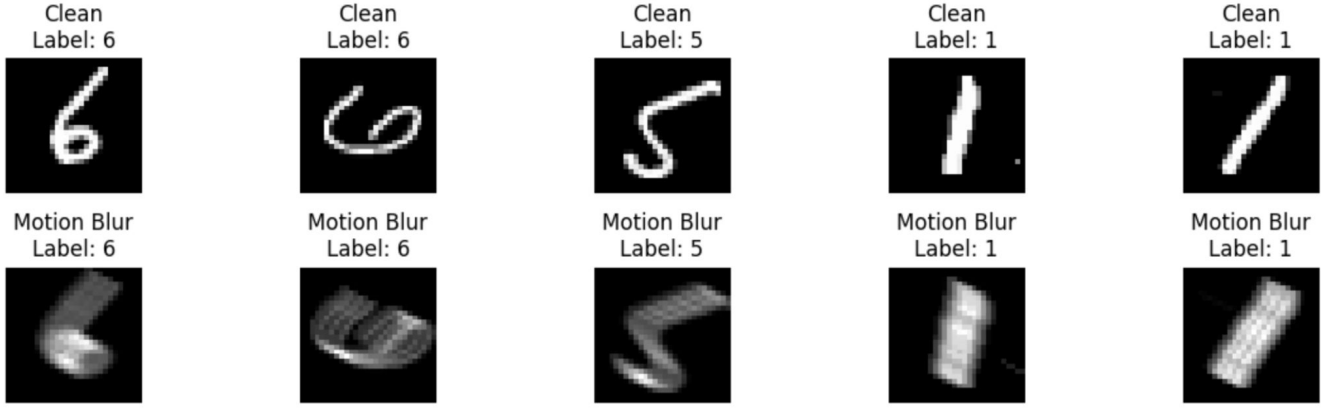


Motion Blur

```python
class AddMotionBlur(BaseEstimator, TransformerMixin):
    def __init__(self, length=5, angle=0):
        #length: number of pixels over which blur is spread (controls blur strength)
        self.length = length
        #angle: direction of the motion blur in degrees (0 = horizontal)
        self.angle = angle

    # Just a placeholder to comply with sklearn,
    # since most transformers need to learn parameters
    def fit(self, X, y=None):
        return self

    def transform(self, X):
        # Create an empty array to store the blurred images
        blurred = np.zeros_like(X)
        for i in range(len(X)):
            img = X[i].reshape(28, 28)
            kernel = self._get_motion_kernel() # Generate motion blur kernel based on length and angle
            blurred_img = convolve(img, kernel, mode='nearest') # Apply motion blur via convolution
            blurred[i] = blurred_img.flatten()
        return np.clip(blurred, 0, 255) # Ensure pixel values stay within valid range [0, 255]
```

```python
    def _get_motion_kernel(self):
        # Create motion blur kernel using specific lengths & angles
        kernel = np.zeros((self.length, self.length)) # initalizes a square kernel filled with zeros
        x_center, y_center = self.length // 2, self.length // 2 # finds the center of the kernel

        # Convert angle to radians and get direction vector
        angle_rad = np.deg2rad(self.angle)
        dx, dy = np.cos(angle_rad), np.sin(angle_rad)

        # Draw line in kernel
        for t in np.linspace(-1, 1, self.length*2):
            x = int(x_center + t * dx * (self.length/2))
            y = int(y_center + t * dy * (self.length/2))
            if 0 <= x < self.length and 0 <= y < self.length: # Make sure the point is inside the kernel
                kernel[y, x] = 1 # Set that location in the kernel to 1 to simulate motion
        return kernel / kernel.sum()  # Normalize
```

Sample Clean vs Motion-Blurred Images
(Length=7, Angle=30°)



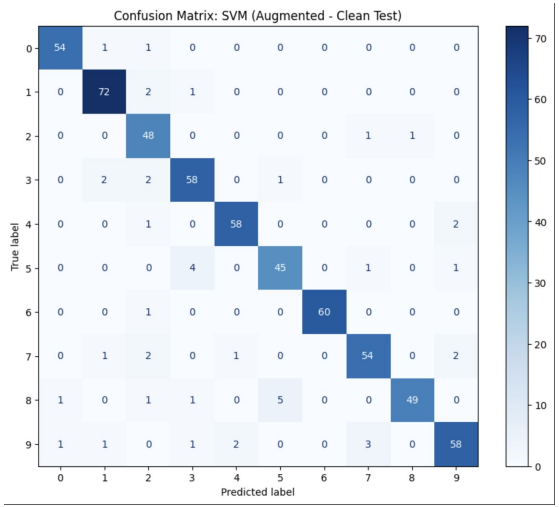
Practical Applications

- Tests how well classifiers deal with moving or shaking imaging

Results - Train on noisy, test on clean

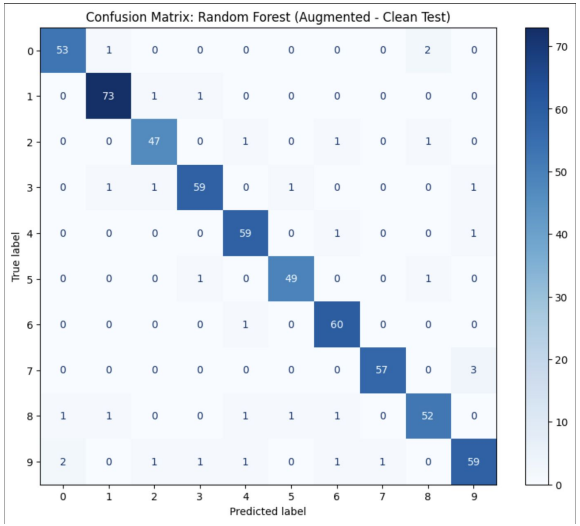
SVM

0.93



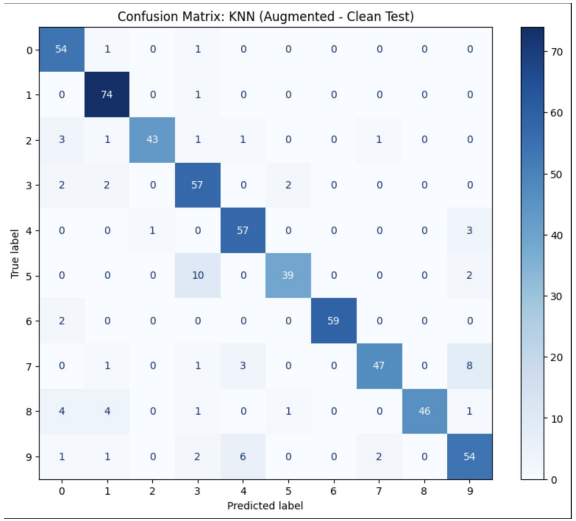
RF

0.95



KNN

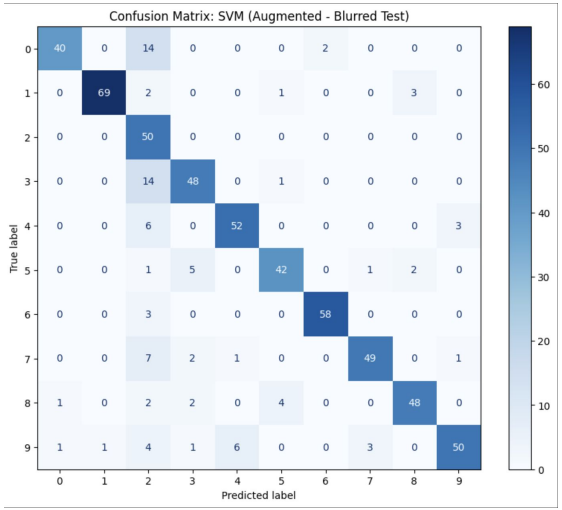
0.88



Results - Train on noisy, test on noisy

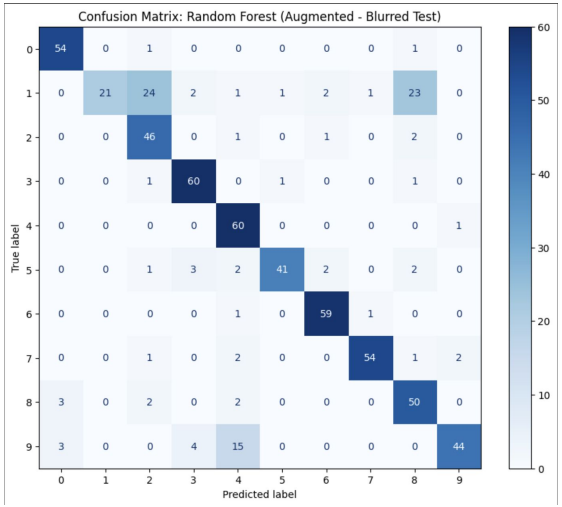
SVM

0.84



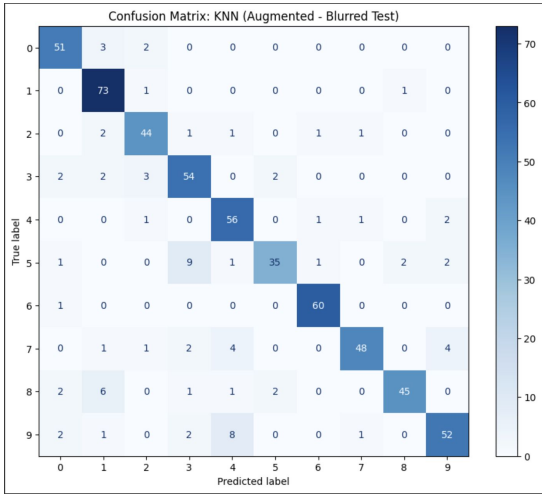
RF

0.81



KNN

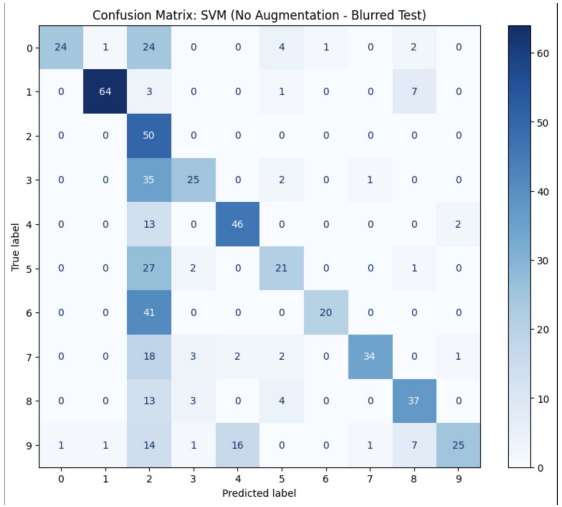
0.86



Results - Train on clean, test on noisy

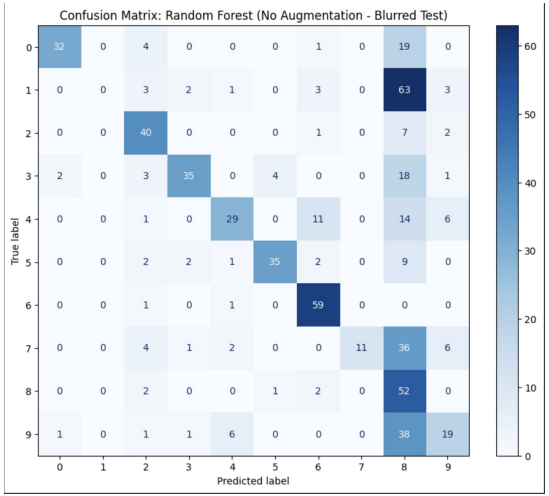
SVM

0.58



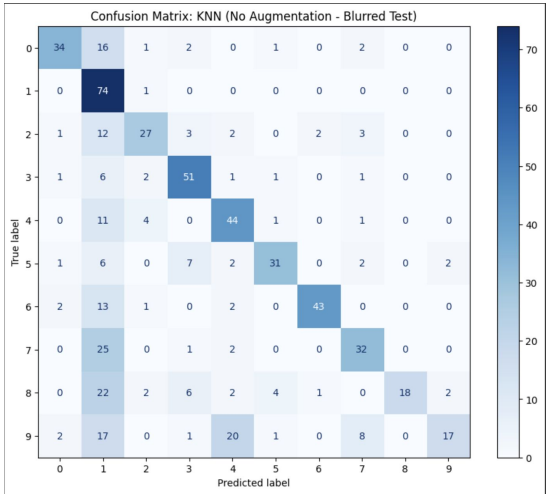
RF

0.52



KNN

0.62

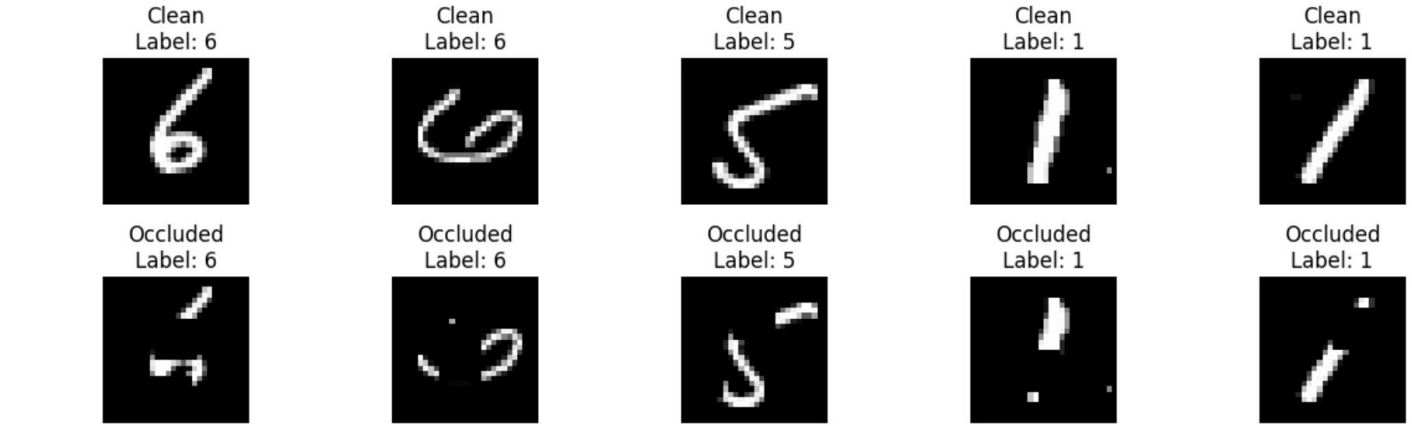


Occlusion

```python
class AddOcclusion(BaseEstimator, TransformerMixin):
    def __init__(self, num_patches=3, patch_size=8):
        self.num_patches = num_patches # Number of patches to apply per image
        self.patch_size = patch_size # Size in pixels of each square patch

    # Just a placeholder to comply with sklearn,
    # since most transformers need to learn parameters
    def fit(self, X, y=None):
        return self

    def transform(self, X):
        # Copy input to avoid modifying original data
        occluded = X.copy()
        for i in range(len(X)):
            img = X[i].reshape(28, 28)
            for _ in range(self.num_patches): # Apply multiple patches
                x = np.random.randint(0, 28 - self.patch_size)
                y = np.random.randint(0, 28 - self.patch_size)
                # Set the pixels in the patch to 0 (black out the region)
                img[y:y+self.patch_size, x:x+self.patch_size] = 0  # Black patches
            occluded[i] = img.flatten()
        return occluded
```

Practical Applications

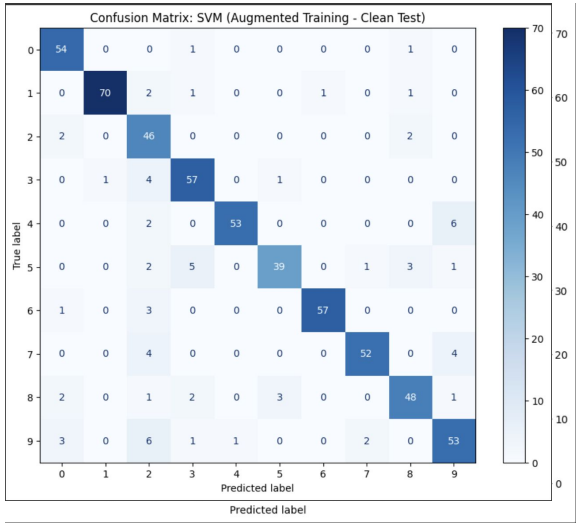
- Tests how well classifiers handle missing or hidden information



Results - train on noisy, test on clean

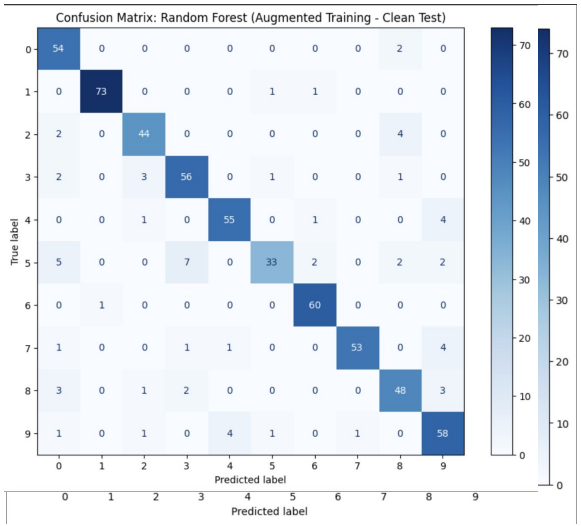
SVM

0.88



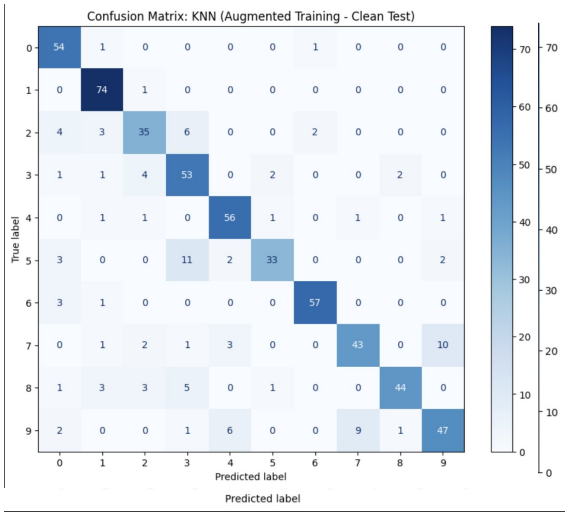
RF

0.89



KNN

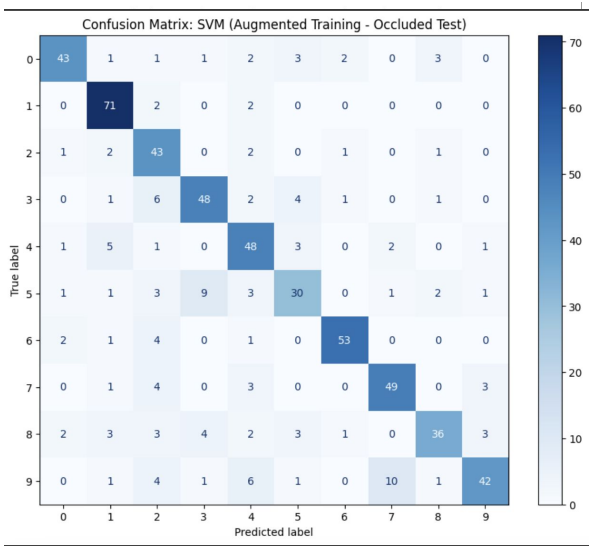
0.83



Results - Train on noisy, test on noisy

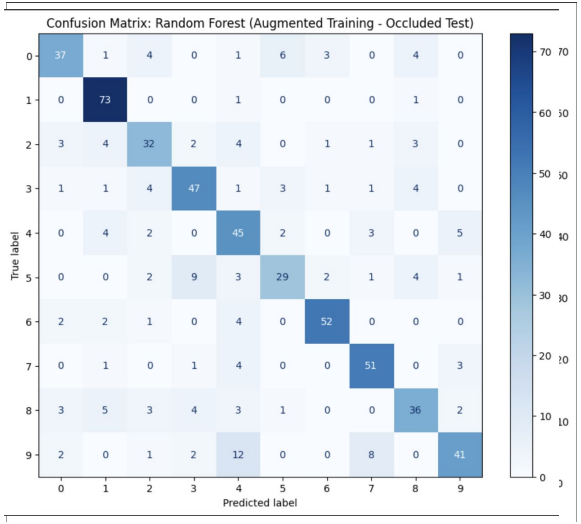
SVM

0.77



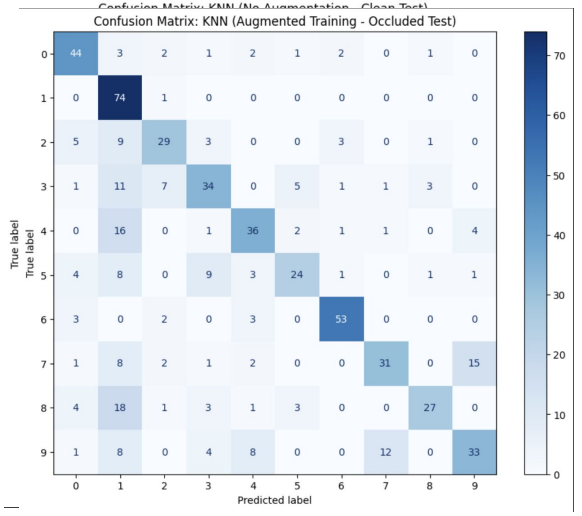
RF

0.74



KNN

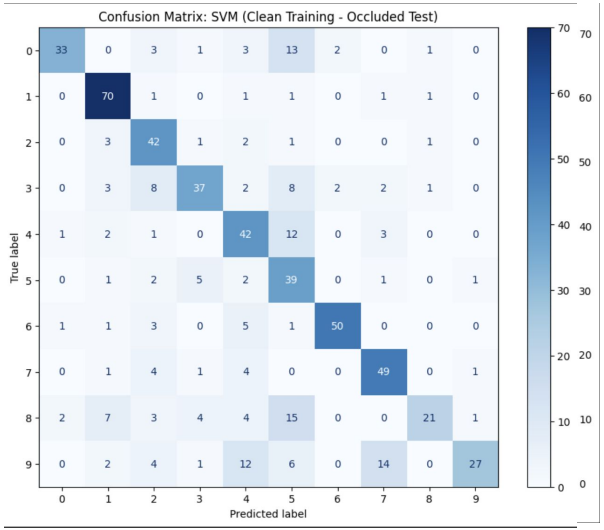
0.64



Results - Test on clean, test on noisy

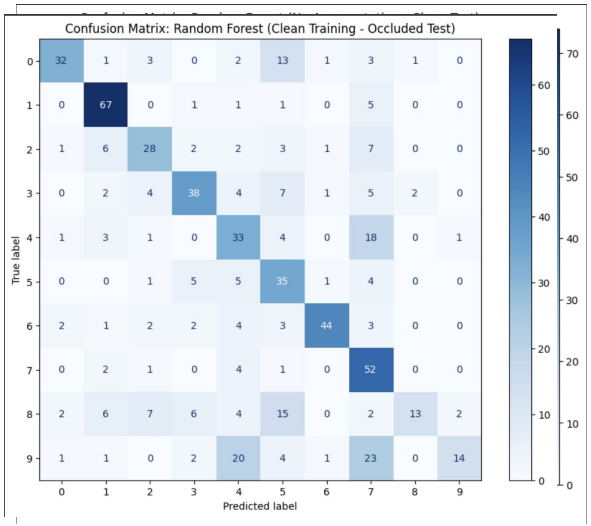
SVM

0.68



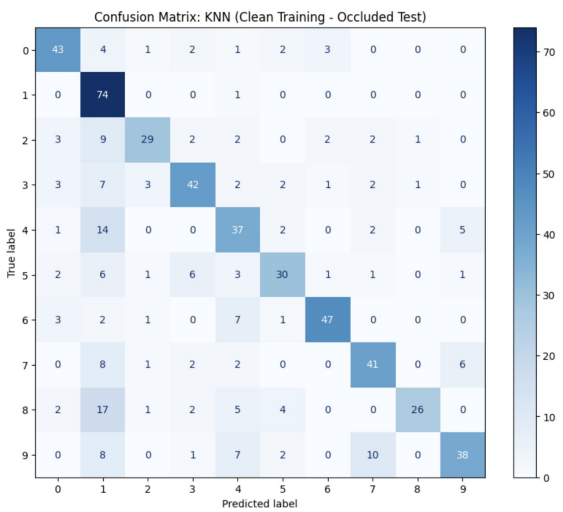
RF

0.59



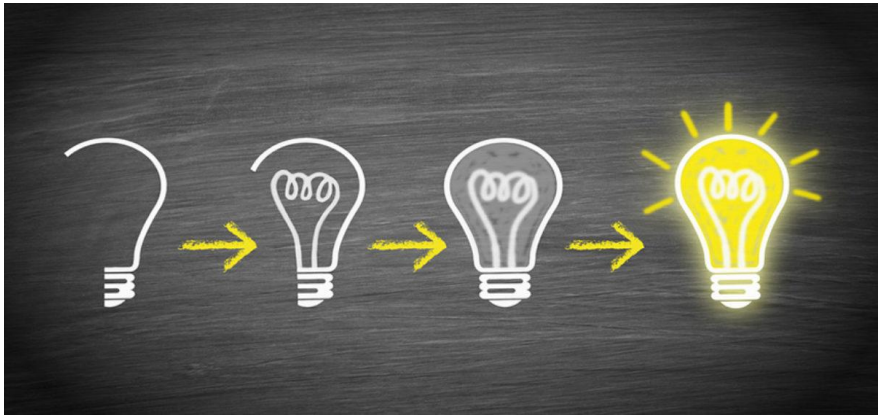
KNN

0.68



What we learned

- How different models responded to noisy vs. clean data, integrating data augmentation into sklearn pipelines using a custom transformer
- Next time- try more models, more noise
- I think it’d be cool to try to get rid of the noise



Expected Vs actual Results (In-general)

- Gaussian White Noise
  - Expected: 1) SVM, 2) RF, 3) KNN
  - Actual: 1) RF, 2) SVM, 3) KNN (mostly) -SVM performed well on noisy/clean but poorly on clean/noisy
- Speckle Noise
  - Expected: 1) SVM, 2) RF, 3) KNN
  - Actual: 1) RF, 2) SVM, 3), KNN
- Motion Blur
  - Expected: 1) SVM, 2) RF, 3) KNN
  - Actual: 1) KNN, 2) RF, 3), SVM
- Occlusion
  - Expected: 1) SVM, 2) RF, 3) KNN
  - Actual: 1) SVM, 2) RF, 2), KNN

References

- The MNIST database of handwritten digits
- https://analyticsindiamag.com/ai-trends/a-guide-to-different-types-of-noises-and-image-denoising-methods/
- https://www.geeksforgeeks.org/noise-models-in-digital-image-processing/
- https://datacarpentry.github.io/image-processing/06-blurring
- ChatGBT - assisted in general information and research + some especially difficult elements of coding