שאלה 1

בשאלה זו נבצע הורדת מימדים על תמונות מ- ***MNIST*** באמצעות ***PCA***.

א. בדומה לתרגיל 4, שאלה 3, השתמשו בקטע הקוד הבא כדי לטעון 8000 תמונות ותוויות מתוך סט הנתונים ***MNIST***:

```
import numpy as np
np.random.seed(42)

from sklearn.datasets import fetch_openml

def fetch_mnist():
    # Download MNIST dataset
    X, y = fetch_openml('mnist_784', version=1, return_X_y=True)
    # Randomly sample 8000 images
    np.random.seed(2)
    indices = np.random.choice(len(X), 8000, replace=False)

    X, y = X[indices], y[indices]
    return X, y

X, y = fetch_mnist()
print(X.shape, y.shape)
```

ב. התאימו ***PCA*** לנתונים שלנו באמצעות:

```
from sklearn.decomposition import PCA

pca = PCA().fit(X)
```

בעת האובייקט ***pca*** מכיל משתנים שימושיים כגון הגורמים הראשיים (***principal components***), ***pca.components_***, וכן השונות המוסברת ע"י כל אחד מהגורמים (ראו פרטים נוספים בתיעוד). לאורך השאלה, כאשר נאמר שונות מוסברת, נתכוון לשונות המוסברת היחסית.

ג. שרטטו את 10 הגורמים הראשיים הראשונים (כתמונות) באמצעות הפונקציה ***plt.imshow*** עם הארגומנט `cmap="binary"`. שימו לב כי יש לשנות את הצורה (***reshape***) של כל גורם בחזרה למימד 28*28 על מנת להציג אותו באמצעות הפונקציה ***imshow***. הסבירו מה התמונות האלה מייצגות.

ד. הציגו גרף של השונות המוסברת ע"י כל גורם ראשי כתלות במספר הגורם, מהגורם ה- 1 עד הגורם ה- 100 בקפיצות של 1. בנוסף, שרטטו גרף של השונות המוסברת המצטברת, כתלות במספר הגורמים. לדוגמה, בגרף הראשון, עבור ערך 20 בציר ה- ***x*** נשרטט בציר ה- ***y*** את השונות המוסברת ע"י הגורם הראשי ה- 20, וכן בגרף השני נשרטט את השונות המוסברת ע"י 20 הגורמים הראשיים הראשונים.

ה. שרטטו שחזורים של חמשת הדוגמאות הראשונות מ- ***X*** עבור שלושה ערכים שונים של שונות מוסברת [50%, 80%, 95%] - סך הכל 15 תמונות. ציינו מעל כל תמונה את התיוג שלה, כמות הגורמים הראשיים על פיהם התבצע השחזור והשונות המוסברת. האם איכות השחזור עולה עם כמות השונות המוסברת? מדוע?

ו. נסמן ב- ***n*** את כמות הגורמים הראשיים הקטנה ביותר שנותנת שונות מוסברת מעל 80% (עליכם לחשב ערך זה). בצעו הורדת מימדים על ***X*** באמצעות ***PCA*** עם ***n*** גורמים ראשיים, נקרא לתוצאה ***X_transformed***. כעת קחו שני מודלים דיפולטיביים של ***sklearn***: ***KNeighborsClassifier(), SVC()***, ואמנו כל מודל פעם אחת על ***X, y*** ופעם שניה על ***X_transformed, y***, כאשר מחלקים אותם לסט אימון וסט מבחן – עם הפקודה הבאה, למשל:

```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

כתבו את שגיאת האימון והמבחן עבור כל ריצה (סה"כ 4 ריצות).
איפה שגיאת המבחן קטנה יותר, עם ***PCA*** או ללא ***PCA***? מדוע?

שאלה 2

בהרצאה ראינו את אלגוריתם ***K-Means***. להלן ניסוח שקול לאלגוריתם עבור מטריקת מרחק אוקלידית:

- ***Input**: $S = \{x_1, \dots, x_m\}$; number of clusters k*
- ***Initialize**: Randomly choose k centroids: $\mu_1, \dots, \mu_k$*
- ***Repeat until convergence**:*
  - $\forall i \in [k]$ ***set*** $C_i = \left\{x \in S : i = \arg\min_j \|x - \mu_j\|\right\}$
    - ***(break ties in an arbitrary manner)***
  - $\forall i \in [k]$ ***update*** $\mu_i = \frac{1}{|C_i|}\sum_{x \in C_i} x$

בנוסף, נגדיר את פונקציית המטרה הבאה:

$$G_{K-Means}(C_1, \dots, C_k) = \min_{\mu_1,\dots,\mu_k} \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2$$

<u>הוכיחו את הטענה הבאה:</u>

בכל איטרציה של אלגוריתם ה ***K-Means***, ערך פונקציית המטרה אינו עולה. כלומר לכל t בריצת האלגוריתם מתקיים:

$$G_{K-Means}\left(C_1^{(t)}, \dots, C_k^{(t)}\right) \le G_{K-Means}\left(C_1^{(t-1)}, \dots, C_k^{(t-1)}\right)$$

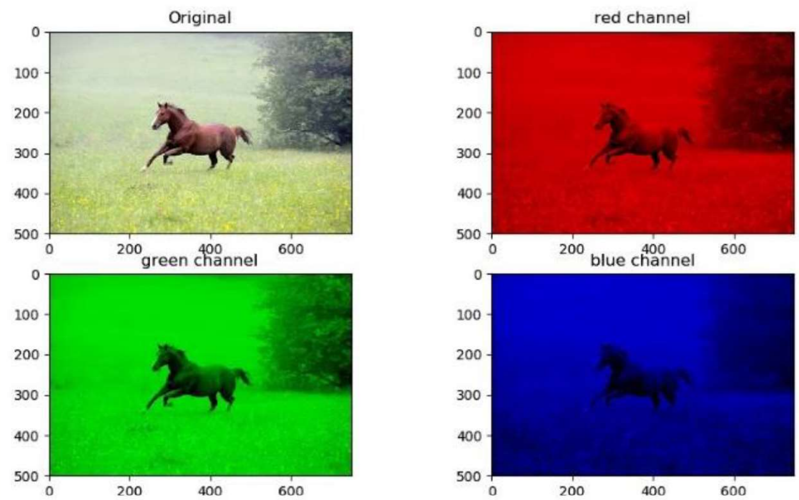
<u>הדרכה:</u>

הגדירו את $\mu(C_i) = \frac{1}{|C_i|}\sum_{x \in C_i} x$ והשתמשו בטענה כי $\mu(C_i) = \arg\min_\mu \sum_{x \in C_i} \|x - \mu\|^2$.

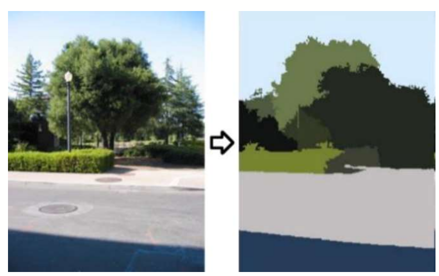
<u>שאלה 3</u>

בשאלה זו נבצע סגמנטציית צבעים בתמונות ע"י שימוש באלגוריתם ***K-Means***.

בתרגיל בית 4, ראינו כי תמונות בשחור לבן מיוצגת ע"י מטריצה דו מימדית של פיקסלים כאשר ערך כל פיקסל נע בין 0-255. על מנת לייצג תמונות צבעוניות, שיטה פופולרית היא לשרשר יחדיו 3 מטריצות דו מימדיות אחת על גבי השניה. כל מטריצה כזו מייצגת את עוצמת ערכי הפיקסלים של ערוץ צבע מסוים: אדום (***R***), ירוק (***G***) וכחול (***B***). לדוגמא:



מכיוון שלפיקסלים סמוכים במרחב יש נטיה להיות בצבעים דומים, משימה אפשרית יכולה להיות הורדת כמות הצבעים בתמונה. מעבר לחסכון בכמות המידע הדרושה בשביל לייצג את התמונה, פתרון משימה זו יכול לסייע לנו להבדיל בצורה אוטומטית בין אובייקטים שונים בתמונה (למשל, שמיים, עצים וקרקע):



בעלום עיבוד התמונה, משימה זו נקראת סגמנטציה מבוססת צבע (***color based segmentation***). בשאלה זו נריץ את אלגוריתם ***K-Means*** על תמונות צבעוניות על מנת למצוא ***K*** צנטרואידים בצבע, ולאחר מכן נחליף כל פיקסל בתמונה בצנטרואיד הצבע שלו. התמונה שתתקבל תכיל בדיוק ***K*** צבעים (חשבו מה מספר הצבעים שניתן לקבל בתמונת ***RGB*** רגילה).

לתרגיל זה מצורפות ארבע תמונות (כלב, חוף, חיפושית וכביש). לכל תמונה נבצע סגמנטציה מבוססת צבע עם אלגוריתם ***K-means*** עבור ערכים שונים של ***K***.

כתבו פונקציה בשם **segement_image_with_kmeans(img, num_colors)** כאשר:

- **img** – מערך **numpy ndarray** של תמונת **RGB** מהצורה **(img_width, img_height, 3)** כאשר **img_width, img_height** הן מידות הרוחב והגובה של התמונה (בפיקסלים).
- **num_colors** – מספר הצבעים (צנטרואידים) – זהו פרמטר **K** עבור אלגוריתם **K-means**.

על הפונקציה לעבוד לפי השלבים הבאים:

1. ביצוע **reshape** של התמונה למערך דו מימדי **X** מהצורה: (**img_width* img_height, 3**). כלומר כל שורה ב**X** מייצגת פיקסל בעל 3 צבעים.

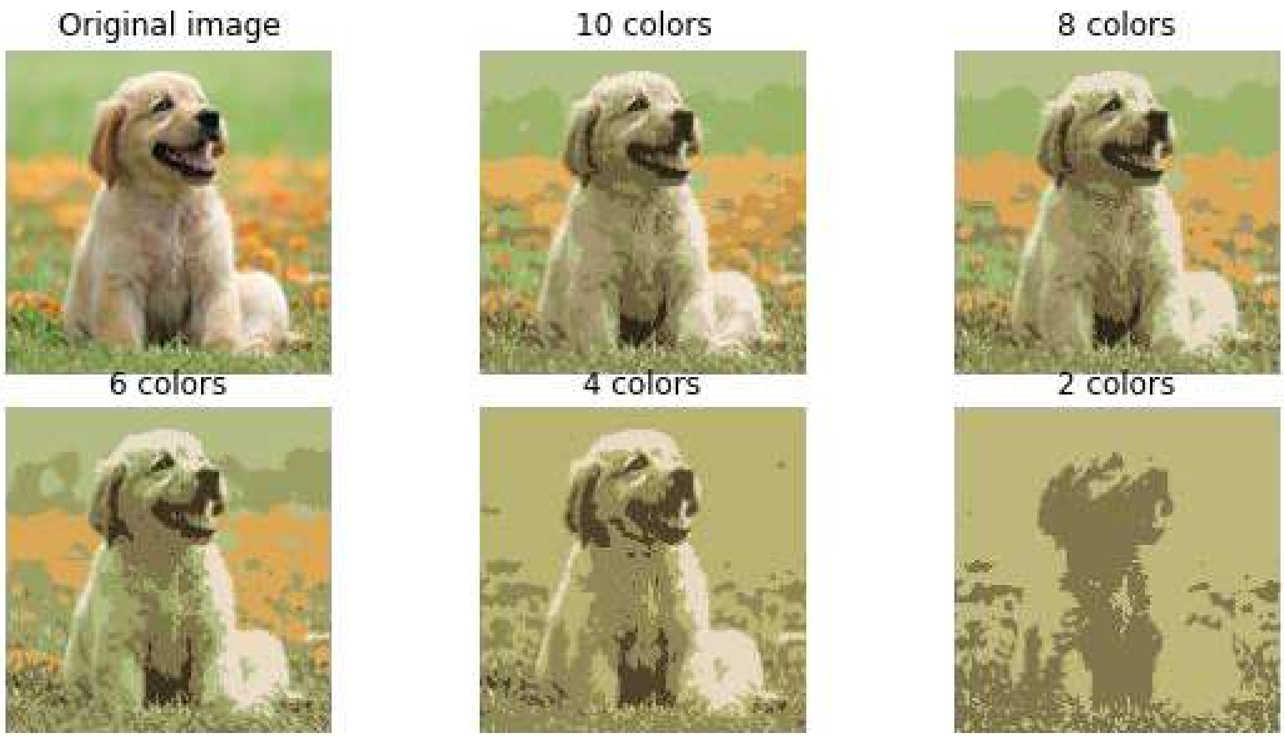
2. התאמת מודל **K-Means** על **X** עם **num_colors** בתור מספר ה **clusters**. יש להשתמש במודל **sklearn.cluster.KMeans** עם **num_colors** בתוך מספר הצנטרואידים ו- **random_state=42**.

3. החלפת כל פיקסל ב **X** בערכי הצנטרואיד שאליו הפיקסל משתייך. יש לשמור את המשתנה בעל הפיקסלים המוחלפים בשם **X_segmented**.

4. ביצוע **reshape** ל-**X_segmented** לתוך משתנה בשם **img_segmented** כך שיהיה במימדים של המשתנה **img** המקורי.

5. החזרה של **img_segmented**.

הריצו את הפונקציה **segement_image_with_kmeans(img, num_colors)** עבור $k \in (\mathbf{10}, \mathbf{8}, \mathbf{6}, \mathbf{4}, \mathbf{2})$ והציגו את התמונה המקורית (**img**) לצד התמונות שעברו סגמנטציית צבע לכל **k** (סה"כ 6 תמונות). הקפידו על כותרת מתאימה לכל פלט (אין צורך לצרף צירים או **ticks** למינהם). פלט תקין לדוגמה:



<u>**הערות חשובות לשאלה:**</u>

- בדיקת השאלה תעשה רק על בסיס הויזואליזציה הסופית של סגמנטציית הצבעים שלכם. באופן כללי, לא נבדוק את כל הקוד שלכם בשאלה הזאת. עם זאת, חובה עליכם לצרף את הקוד המלא שכתבתם. יש לצרף את הקוד **כטקסט הניתן להעתקה** מתוך קובץ הpdf (**ולא כצילום מסך או כקובץ נפרד**).
- אם יתעורר חשד שהויזואליזציות שצירפתם אינן מגיעות מהקוד שצירפתם, נריץ את הקוד שצירפתם. לפיכך, במידה ואחד או יותר מהתנאים הבאים יתקיימו, **השאלה תספוג הורדת נקודות משמעותית**:
    - השאלה תוגש בלי קוד מצורף
    - לשאלה יצורף קוד שלא ניתן להעתיק אותו (צילום מסך) או קובץ נפרד (למשל קובץ **py**.)
    - הקוד המצורף אינו קריא (למשל: לא מתועד, שמות לא משמעותיים למשתנים, קוד מסובך וארוך יתר על המידה)
    - הקוד המצורף לא רץ
    - הקוד המצורף רץ בזמן לא סביר (מעל 2 דקות לתמונה לכל ערכי ה-**k**)
    - הקוד המצורף מייצר ויזואליזציות שונות מהותית מאלו שצירפתם

Question 1.1

In [1]:
```
import numpy as np
np.random.seed(42)
from sklearn.datasets import fetch_openml
import matplotlib.pyplot as plt
```

In [2]:
```
def fetch_mnist():
    # Download MNIST dataset
    X, y = fetch_openml('mnist_784', version=1, return_X_y=True)
    # Randomly sample 8000 images
    np.random.seed(2)
    indices = np.random.choice(len(X), 8000, replace=False)

    X, y = X[indices], y[indices]
    return X, y

X, y = fetch_mnist()
print(X.shape, y.shape)
```

```
(8000, 784) (8000,)
```

Question 1.2

In [3]:
```
from sklearn.decomposition import PCA
pca = PCA().fit(X)
```

In [4]:
```
pca.components_
```

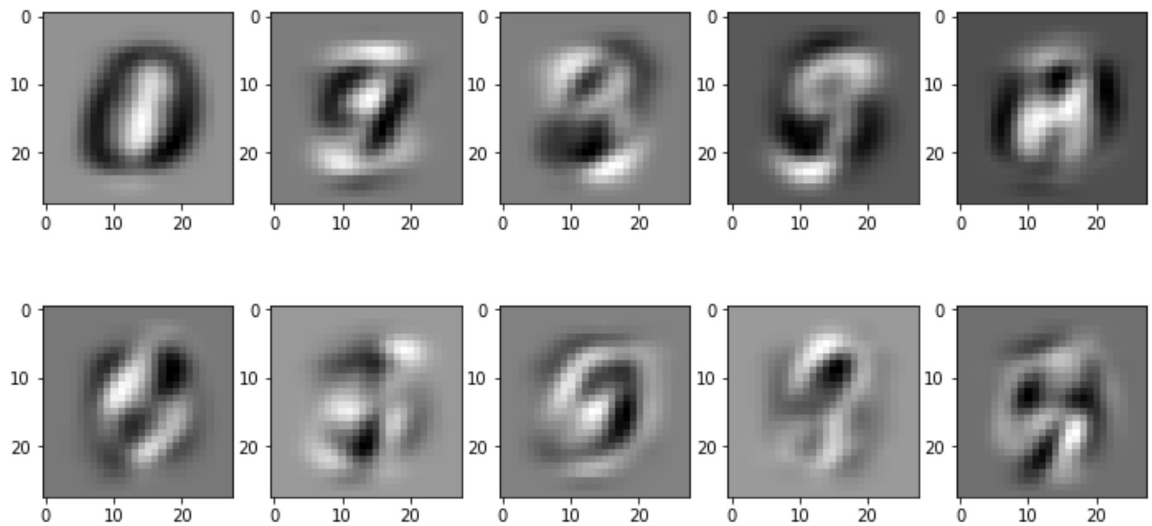
Out[4]:
```
array([[-4.69337428e-20,  2.42861287e-17, -8.32667268e-17, ...,
        -0.00000000e+00, -0.00000000e+00, -0.00000000e+00],
       [-1.40435167e-19, -7.63278329e-17, -1.11022302e-16, ...,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00],
       [-2.91207566e-19,  1.38777878e-17, -5.55111512e-17, ...,
        -0.00000000e+00, -0.00000000e+00, -0.00000000e+00],
       ...,
       [ 0.00000000e+00, -8.04613903e-02,  2.59629077e-01, ...,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00],
       [-0.00000000e+00,  2.95100438e-02,  1.86784874e-02, ...,
        -0.00000000e+00, -0.00000000e+00, -0.00000000e+00],
       [ 0.00000000e+00, -1.15071909e-01,  1.03930147e-01, ...,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00]])
```

Question 1.3

In [5]:
```
def show_first_10():
    k=10
    X_k = np.reshape(pca.components_[:k], (k,28,28))

    fig = plt.figure(figsize=(12,6))
    columns=5
    rows=2
    ax=[]
    for i in range(columns*rows):
        # create subplot and append to ax
        ax.append(fig.add_subplot(rows,columns,i+1))
        plt.imshow(X_k[i],cmap="binary")
    plt.show()

show_first_10()
```

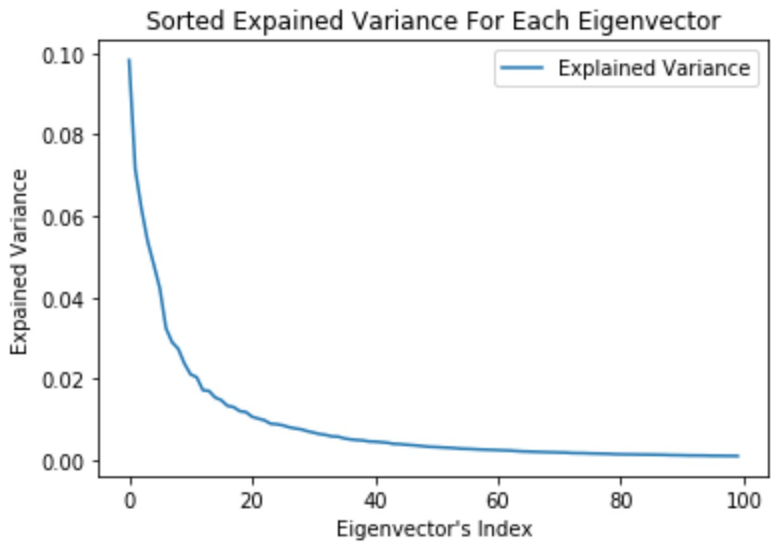


The images represent the eigenvectors corresponding to the largest 10 eigenvalues.

Question 1.4

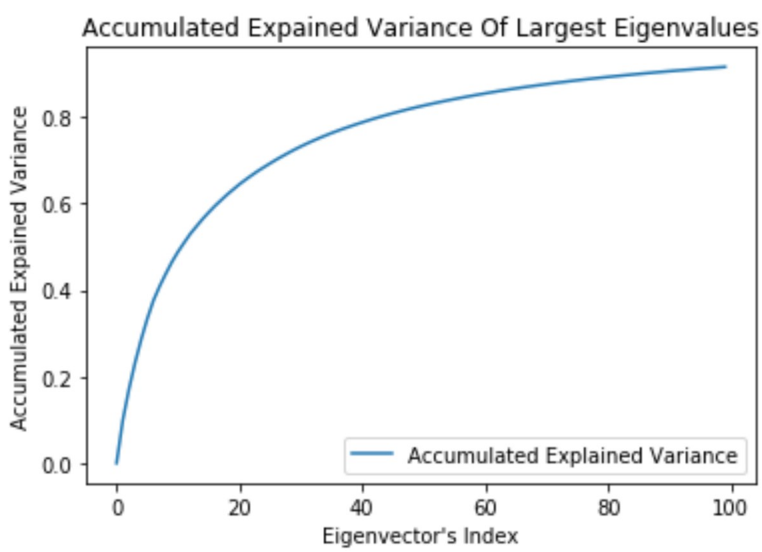
In [6]:
```
plt.plot([m for m in range(100)], pca.explained_variance_ratio_[:100],
label="Explained Variance")
plt.title("Sorted Expained Variance For Each Eigenvector")
plt.legend()
plt.xlabel("Eigenvector's Index")
plt.ylabel("Expained Variance")
```

Out[6]: `Text(0, 0.5, 'Expained Variance')`



In [7]:
```
plt.plot([m for m in range(100)], [pca.explained_variance_ratio_[:i].su
m() for i in range(100)],  label="Accumulated Explained Variance")
plt.title("Accumulated Expained Variance Of Largest Eigenvalues")
plt.legend()
plt.xlabel("Eigenvector's Index")
plt.ylabel("Accumulated Expained Variance")
```

Out[7]: `Text(0, 0.5, 'Accumulated Expained Variance')`



Question 1.5

In [8]:
```
def get_n_with_var(v=1):
    for i in range(pca.n_components_):
        if pca.explained_variance_ratio_[:i].sum() >= v:
            return i
    return pca.n_components_
```

In [9]:
```
get_n_with_var(0.5)
```

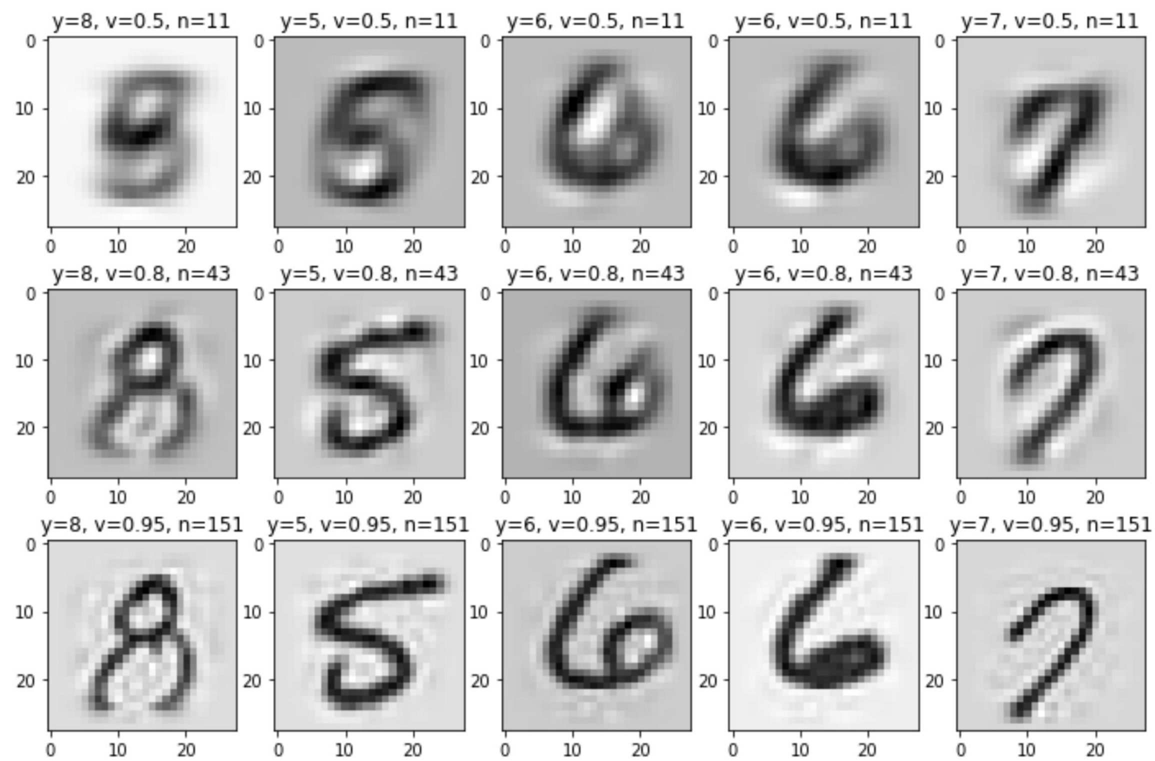
Out[9]: 11

In [10]:
```
def get_title(i):
    if i in range(0,5):
        v = 0.5
    elif i in range(5,10):
        v = 0.8
    else:
        v = 0.95
    n = get_n_with_var(v)
    s = "y={}, v={}, n={}".format(y[i%5],v,n)
    return s
```

In [11]:
```
X_tranformed = np.array([])
for v in [0.5, 0.8, 0.95]:
    pca = PCA(n_components=v, svd_solver='full')
    pca.fit(X)
    X_5 = X[:5]
    X_5 = pca.transform(X_5)
    X_5 = pca.inverse_transform(X_5)
    X_tranformed = np.append(X_tranformed, X_5)

k=15
X_k = np.reshape(X_tranformed, (k,28,28))
fig = plt.figure(figsize=(12,8))
columns=5
rows=3
ax=[]

for i in range(columns*rows):
    # create subplot and append to ax
    ax.append(fig.add_subplot(rows,columns,i+1))
    plt.imshow(X_k[i],cmap="binary")
    plt.title(get_title(i))
plt.show()
```



The quality of the reconstruction get better as we increase the explained variance, this observation fits our expectation: As we saw in the lecture, increasing the explained variance is decreasing the reconstruction error, meaning we are able to reconstruct the original data more accurately.

Question 1.6

In [12]:
```
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
```

In [13]:
```
pca = PCA(n_components=0.8, svd_solver='full').fit(X)
X_tranformed = pca.transform(X)

svc = SVC()
knn = KNeighborsClassifier()
svc_t = SVC()
knn_t = KNeighborsClassifier()

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,rand
om_state=42)
X_train_t,X_test_t,y_train_t,y_test_t = train_test_split(X_tranformed,
y,test_size=0.3,random_state=42)

svc.fit(X_train, y_train)
knn.fit(X_train, y_train)
svc_t.fit(X_train_t, y_train_t)
knn_t.fit(X_train_t, y_train_t)

svc_train_score = svc.score(X_train, y_train)
svc_test_score = svc.score(X_test, y_test)
knn_train_score = knn.score(X_train, y_train)
knn_test_score = knn.score(X_test, y_test)

svc_t_train_score = svc_t.score(X_train_t, y_train_t)
svc_t_test_score = svc_t.score(X_test_t, y_test_t)
knn_t_train_score = knn_t.score(X_train_t, y_train_t)
knn_t_test_score = knn_t.score(X_test_t, y_test_t)

print("svc train error rate:", 1-svc_train_score)
print("svc test error rate:", 1-svc_test_score)
print("svc_transforemed train error rate:", 1-svc_t_train_score)
print("svc_transforemed test error rate:", 1-svc_t_test_score)

print("knn train error rate:", 1-knn_train_score)
print("knn test error rate:", 1-knn_test_score)
print("knn_transforemed train error rate:", 1-knn_t_train_score)
print("knn_transforemed test error rate:", 1-knn_t_test_score)
```

```
svc train error rate: 0.015000000000000013
svc test error rate: 0.04500000000000004
svc_transforemed train error rate: 0.013214285714285734
svc_transforemed test error rate: 0.03749999999999998
knn train error rate: 0.043749999999999956
knn test error rate: 0.060833333333333295
knn_transforemed train error rate: 0.038214285714285756
knn_transforemed test error rate: 0.050833333333333286
```

We notice that the test error of the models using PCA is smaller than those which does not. Curse of dimensionality could provide a possible explantation for the above results. When using PCA we reduce the dimensoinality of the data and presumably reduce some of the noice. Models such as KNN and SVC are highly affected by high dimensional data since they use distance to classify, therefore PCA improves the test results when reducing the dimensions.

Question 3

In [14]:
```
def get_title(i):
    if (i+1)%6!=0:
        return str(2*((i+1)%6))+" colors"
    else:
        return "Original Image"
```

In [15]:
```
from sklearn.cluster import KMeans
def segement_image_with_kmeans(img, num_colors):
    img_width, img_height, temp = img.shape
    X = np.reshape(img, (img_width*img_height, 3))

    knn = KMeans(n_clusters=num_colors, random_state=42).fit(X)
    X_segmented = np.array([knn.cluster_centers_[i] for i in knn.predic
t(X)])
    img_segmented = np.reshape(X_segmented, (img_width, img_height, 3))

    return img_segmented
```

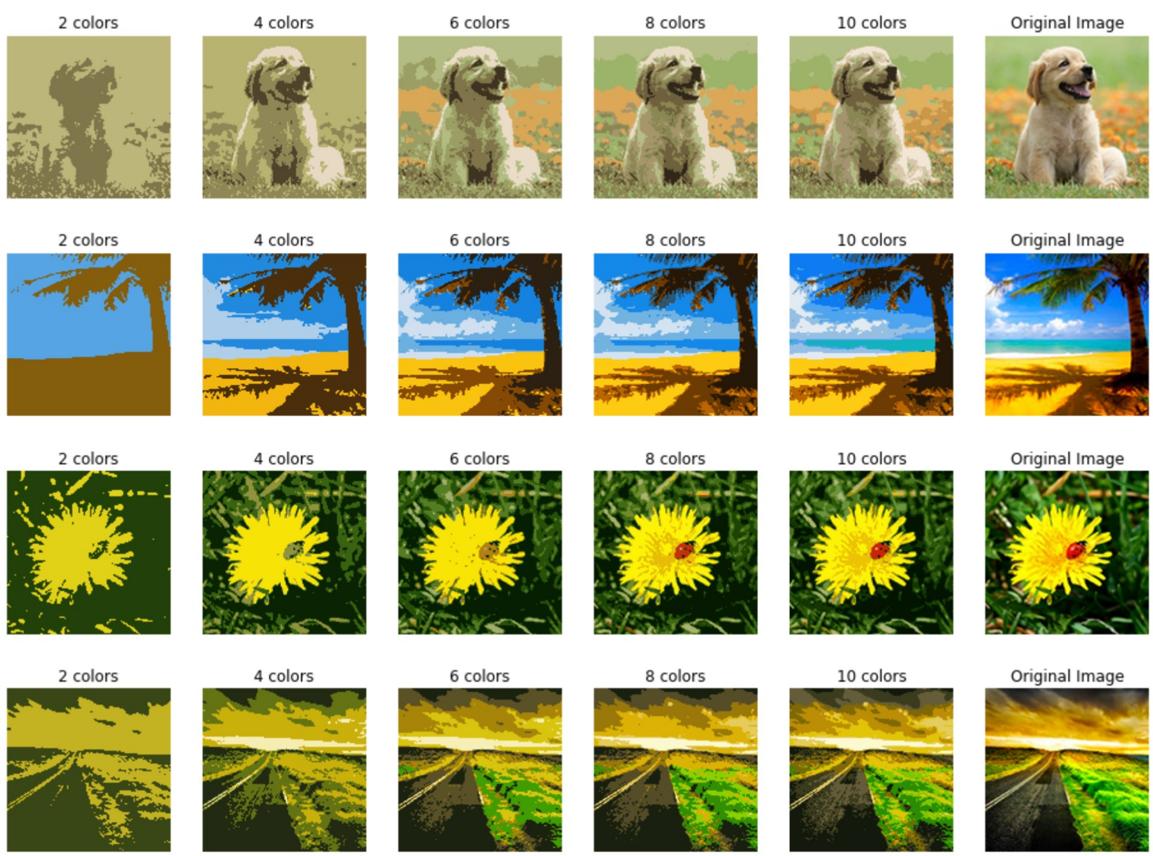
In [16]:

```python
from matplotlib.image import imread
imgs = ['dog.png', 'beach.png', 'ladybug.png', 'road.png']
K = [j for j in range(2,12,2)]
transfomed_imgs = np.array([])

for i in imgs:
    for k in K:
        img = imread(i)
        img_segmented = segement_image_with_kmeans(img, k)
        transfomed_imgs = np.append(transfomed_imgs, img_segmented)
    transfomed_imgs = np.append(transfomed_imgs, np.reshape(img, (128,128,3)))

columns=6
rows=4
k=columns*rows
X_k = np.reshape(transfomed_imgs, (k,128,128,3))
fig = plt.figure(figsize=(16,12))
ax=[]

for i in range(columns*rows):
    # create subplot and append to ax
    ax.append(fig.add_subplot(rows,columns,i+1))
    plt.imshow(X_k[i],cmap="binary")
    plt.axis('off')
    plt.title(get_title(i))
plt.show()
```



שאלה 2

תהא $C_k^{(t-1)}, \dots, C_1^{(t-1)}$ החלוקה לקבוצות של הנקודות.

תהא $v^{(t-1)} = G_{K\ Means}(\ C_1^{(t-1)}, \dots, C_k^{(t-1)})$ ערך פונקציית המטרה בשלב t-1.

נשים לי כי מתקיים בכל איטרציה של האלגוריתם:

ראשית, בכל איטרציה t, אנו משייכים כל תצפית לקלאסטר עם הסנטרואיד שמרחקו הוא המינימלי מבין כל המרחקים.

- ← הביטוי $||x - \mu_i||$ המופיע בפונקציית המטרה ואותו אנו מנסים למזער לא גדל בפעולה זו לכל *x*.
- ← ערך פונקציית המטרה $v^{(t)}$ שהיא סכום מרחקים לא גדלה בעת ביצוע פעולה זו.

כפי שמופיע בהדרכה מתקיים כי $\mu_i(C_i) = argmin_\mu \sum_{x \in C_i} ||x - \mu_i||$

שנית, באיטרציה ה-*t* נקבע כל סנטרואיד להיות הממוצע של הנקודות השייכות לקלאסטר.

- ← מצירוף הטענות לאחר עדכון המרכזים נקבל כי פונקציית המטרה $v^{(t)}$ הסוכמת את מרחקי הנקודות מהמרכזים לא גדלה מפני שכל מרכז הוגדר להיות הנקודה שממזערת את סכום המרחקים מהנקודות ששייכות לקבוצה.

הסתיימה האיטרציה *t*, נעבור לאיטרציה *t+1*.

- ← ערך פונקציית המטרה לא גדל באף שלב באיטרציה ולכן מתקיים כי:

$$G_{K-Means}\left(C_1^{(t-1)}, \dots, C_k^{(t-1)}\right) \le G_{K-Means}\left(C_1^{(t)}, \dots, C_k^{(t)}\right)$$