Loading the packages

In [11]:
```
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt

# scikit-learn functions
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.decomposition import PCA, NMF

# TensorFlow / Keras functions
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, MaxPooling2D, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.datasets import cifar10
```

Loading the CIFAR-10 data

In [19]:
```
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

print('X shapes: ', X_train.shape, X_test.shape)
print('y shapes: ', y_train.shape, y_test.shape)
```

```
X shapes:  (50000, 32, 32, 3) (10000, 32, 32, 3)
y shapes:  (50000, 1) (10000, 1)
```

EDA and Preprocessing

In [20]:
```python
# labels: ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'shi
p', 'truck']
# Filter training and test sets to 2 images
frog = 6
ship = 8
train_ind = np.where((y_train == frog) | (y_train == ship))[0]
test_ind = np.where((y_test == frog) | (y_test == ship))[0]
y_train = y_train[train_ind]
X_train = X_train[train_ind]
y_test = y_test[test_ind]
X_test = X_test[test_ind]

# Relabel frog as 0 and ship as 1 for binary classification
y_train[y_train == frog] = 0
y_train[y_train == ship] = 1
y_test[y_test == frog] = 0
y_test[y_test == ship] = 1

# Create validation set
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=.5, st
ratify=y_train, random_state=1)

# Printing the shapes
print('X shapes: ', X_train.shape, X_valid.shape, X_test.shape)
print('y shapes: ', y_train.shape, y_valid.shape, y_test.shape)
```

```
X shapes:  (5000, 32, 32, 3) (5000, 32, 32, 3) (2000, 32, 32, 3)
y shapes:  (5000, 1) (5000, 1) (2000, 1)
```

In [36]:
```python
# Reshaping the data
X_train = X_train.reshape(X_train.shape[0], -1)
X_valid = X_valid.reshape(X_valid.shape[0], -1)
X_test = X_test.reshape(X_test.shape[0], -1)

# reshaping the output parameter to 1-D array
y_train = y_train.ravel()

print('X shapes: ', X_train.shape, X_valid.shape, X_test.shape)
print('y shapes: ', y_train.shape, y_valid.shape, y_test.shape)
```

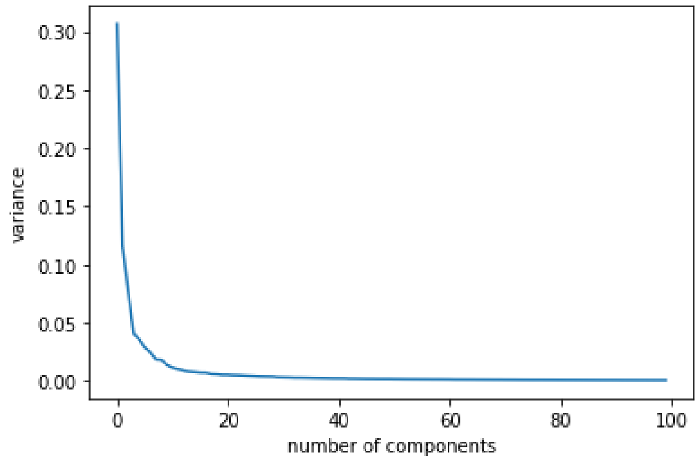
```
X shapes:  (5000, 3072) (5000, 3072) (2000, 3072)
y shapes:  (5000,) (5000, 1) (2000, 1)
```

In [37]:
```python
# Standardizing the data
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_valid = scaler.transform(X_valid)
X_test = scaler.transform(X_test)
```

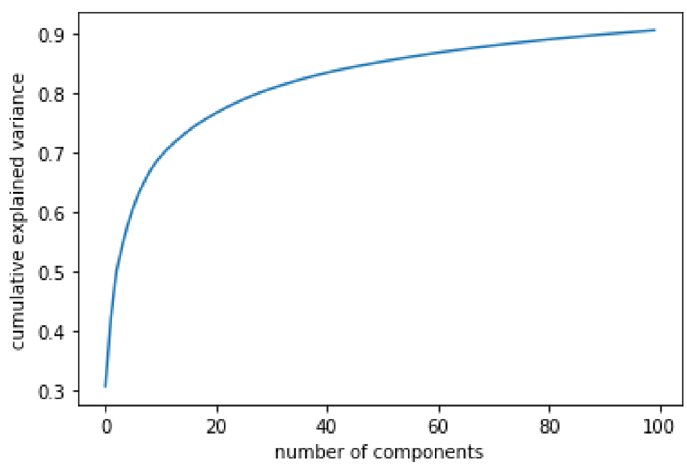
In [38]:
```python
# Applying Demention Reduction PCA
pca = PCA(n_components=100, random_state=1)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)
X_valid = pca.transform(X_valid)
print('X shapes: ', X_train.shape, X_valid.shape, X_test.shape)
```

```
X shapes:  (5000, 100) (5000, 100) (2000, 100)
```

In [17]:
```
# Scree plot of the principal components (explained variance vs. number of components)
plt.plot(pca.explained_variance_ratio_)
plt.xlabel('number of components')
plt.ylabel('variance')
plt.show()
```



In [18]:
```
# Scree plot of cumulative explained variance vs. number of components
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.show()
```



According to the above graphs with 100 components we will have %90 of variation explained. So will use 100 components for the rest of this project.

Initial KNN

In [39]:
```
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
```

Out[39]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

In [40]:
```
preds_knn = knn.predict(X_test)
print('KNN accuracy: {:.3f}'.format(accuracy_score(y_test, preds_knn)))
```

```
KNN accuracy: 0.911
```

Optimized KNN

In [9]:
```
knn = KNeighborsClassifier()
ks = np.arange(2, 7)
gs = GridSearchCV(knn, param_grid={'n_neighbors':ks}, scoring='accuracy', cv = 5)
gs.fit(X_train, y_train)
```

Out[9]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
                                            metric='minkowski',
                                            metric_params=None, n_jobs=None,
                                            n_neighbors=5, p=2,
                                            weights='uniform'),
             iid='warn', n_jobs=None,
             param_grid={'n_neighbors': array([2, 3, 4, 5, 6])},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=0)
```

In [10]:
```
gs.best_params_
```

Out[10]:
```
{'n_neighbors': 3}
```

In [11]:
```
preds = gs.predict(X_test)
print('KNN accuracy: {:.3f}'.format(accuracy_score(y_test, preds)))
```

```
KNN accuracy: 0.914
```

Initial Random Forest

In [20]:
```
rf = RandomForestClassifier(n_estimators  = 100)
rf.fit(X_train, y_train)
```

Out[20]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [21]:
```
preds = rf.predict(X_test)
print('Random Forest accuracy: {:.3f}'.format(accuracy_score(y_test, preds)))
```

```
Random Forest accuracy: 0.922
```

Optimized Random Forest

In [22]:
```
rf = RandomForestClassifier()
trees = [100, 200, 300]
depths = [6, 7, 8]
gs = GridSearchCV(rf, param_grid={'n_estimators':trees, 'max_depth':depths},
scoring='accuracy', cv = 5)
gs.fit(X_train, y_train)
```

Out[22]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=RandomForestClassifier(bootstrap=True, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators='warn', n_jobs=None,
                                              oob_score=False,
                                              random_state=None, verbose=0,
                                              warm_start=False),
             iid='warn', n_jobs=None,
             param_grid={'max_depth': [6, 7, 8],
                         'n_estimators': [100, 200, 300]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=0)
```

In [23]:
```
gs.best_params_
```

Out[23]:
```
{'max_depth': 8, 'n_estimators': 300}
```

In [16]:
```
preds = gs.predict(X_test)
print('Random Forest accuracy: {:.3f}'.format(accuracy_score(y_test, preds)))
```

```
Random Forest accuracy: 0.924
```

Initial Gradient Boosted

In [17]:
```
gb = GradientBoostingClassifier()
gb.fit(X_train, y_train)
```

Out[17]:
```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_iter_no_change=None, presort='auto',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

In [18]:
```
preds = gb.predict(X_test)
print('Gradient Boosting accuracy: {:.3f}'.format(accuracy_score(y_test, preds)))
```

```
Gradient Boosting accuracy: 0.928
```

Optimized Gradient Boosted

In [19]:
```
gb = GradientBoostingClassifier()
trees = [200, 300]
learning = [.05, .1, .2]
gs = GridSearchCV(gb, param_grid={'n_estimators':trees, 'learning_rate':learning},
scoring='accuracy', cv = 5)
gs.fit(X_train, y_train)
```

Out[19]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=GradientBoostingClassifier(criterion='friedman_mse',
                                                  init=None, learning_rate=0.1,
                                                  loss='deviance', max_depth=3,
                                                  max_features=None,
                                                  max_leaf_nodes=None,
                                                  min_impurity_decrease=0.0,
                                                  min_impurity_split=None,
                                                  min_samples_leaf=1,
                                                  min_samples_split=2,
                                                  min_weight_fraction_leaf=0.0,
                                                  n_estimators=100,
                                                  n_iter_no_change=None,
                                                  presort='auto',
                                                  random_state=None,
                                                  subsample=1.0, tol=0.0001,
                                                  validation_fraction=0.1,
                                                  verbose=0, warm_start=False),
             iid='warn', n_jobs=None,
             param_grid={'learning_rate': [0.05, 0.1, 0.2],
                         'n_estimators': [200, 300]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=0)
```

In [20]:
```
gs.best_params_
```

Out[20]:
```
{'learning_rate': 0.1, 'n_estimators': 300}
```

In [21]:
```
preds = gs.predict(X_test)
print('Gradient Boosting accuracy: {:.3f}'.format(accuracy_score(y_test, preds)))
```

```
Gradient Boosting accuracy: 0.933
```

Changing the shape of y_train for Neural Net

In [38]:
```
y_train = y_train.reshape(-1, 1)
```

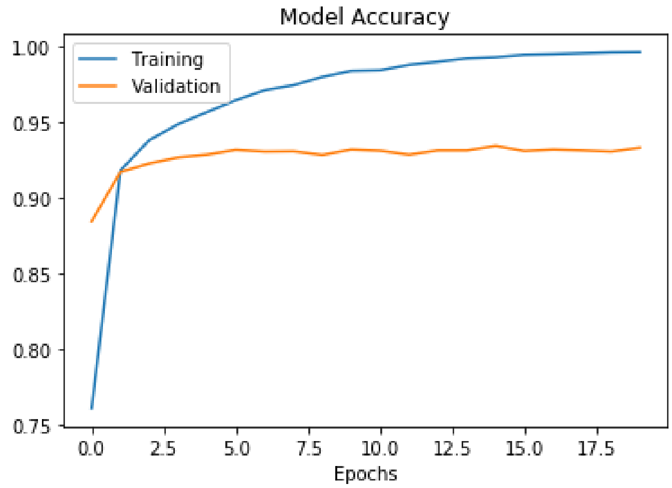
Initial Neural Net

In [39]:
```
model1 = Sequential()
model1.add(Dense(50, activation='relu', input_shape=(X_train.shape[1], )))
model1.add(Dense(10, activation='relu'))
model1.add(Dense(10, activation='relu'))
model1.add(Dense(1, activation='sigmoid'))

model1.compile(optimizer='adam',
               loss='binary_crossentropy',
               metrics=['accuracy'])

model1.fit(X_train, y_train, epochs=20, batch_size=50,
           validation_data=(X_valid, y_valid), verbose = 2)

model1.evaluate(X_test, y_test, verbose = 2)
```

```
Train on 5000 samples, validate on 5000 samples
Epoch 1/20
5000/5000 - 1s - loss: 0.5289 - accuracy: 0.7608 - val_loss: 0.2885 - val_accuracy: 0.88
44
Epoch 2/20
5000/5000 - 0s - loss: 0.2202 - accuracy: 0.9180 - val_loss: 0.2215 - val_accuracy: 0.91
70
Epoch 3/20
5000/5000 - 0s - loss: 0.1659 - accuracy: 0.9380 - val_loss: 0.2120 - val_accuracy: 0.92
26
Epoch 4/20
5000/5000 - 0s - loss: 0.1421 - accuracy: 0.9486 - val_loss: 0.2036 - val_accuracy: 0.92
66
Epoch 5/20
5000/5000 - 0s - loss: 0.1216 - accuracy: 0.9566 - val_loss: 0.2028 - val_accuracy: 0.92
84
Epoch 6/20
5000/5000 - 0s - loss: 0.1060 - accuracy: 0.9644 - val_loss: 0.2043 - val_accuracy: 0.93
16
Epoch 7/20
5000/5000 - 0s - loss: 0.0922 - accuracy: 0.9710 - val_loss: 0.2078 - val_accuracy: 0.93
04
Epoch 8/20
5000/5000 - 0s - loss: 0.0814 - accuracy: 0.9744 - val_loss: 0.2140 - val_accuracy: 0.93
06
Epoch 9/20
5000/5000 - 0s - loss: 0.0712 - accuracy: 0.9798 - val_loss: 0.2082 - val_accuracy: 0.92
82
Epoch 10/20
5000/5000 - 0s - loss: 0.0609 - accuracy: 0.9838 - val_loss: 0.2135 - val_accuracy: 0.93
18
Epoch 11/20
5000/5000 - 1s - loss: 0.0528 - accuracy: 0.9842 - val_loss: 0.2185 - val_accuracy: 0.93
10
Epoch 12/20
5000/5000 - 0s - loss: 0.0451 - accuracy: 0.9878 - val_loss: 0.2298 - val_accuracy: 0.92
84
Epoch 13/20
5000/5000 - 1s - loss: 0.0382 - accuracy: 0.9898 - val_loss: 0.2328 - val_accuracy: 0.93
12
Epoch 14/20
5000/5000 - 0s - loss: 0.0325 - accuracy: 0.9920 - val_loss: 0.2426 - val_accuracy: 0.93
12
Epoch 15/20
5000/5000 - 1s - loss: 0.0275 - accuracy: 0.9928 - val_loss: 0.2486 - val_accuracy: 0.93
42
Epoch 16/20
5000/5000 - 0s - loss: 0.0227 - accuracy: 0.9944 - val_loss: 0.2652 - val_accuracy: 0.93
08
Epoch 17/20
5000/5000 - 1s - loss: 0.0193 - accuracy: 0.9948 - val_loss: 0.2719 - val_accuracy: 0.93
18
Epoch 18/20
5000/5000 - 1s - loss: 0.0167 - accuracy: 0.9954 - val_loss: 0.2883 - val_accuracy: 0.93
12
Epoch 19/20
5000/5000 - 1s - loss: 0.0140 - accuracy: 0.9960 - val_loss: 0.2897 - val_accuracy: 0.93
04
Epoch 20/20
5000/5000 - 1s - loss: 0.0110 - accuracy: 0.9962 - val_loss: 0.3009 - val_accuracy: 0.93
30
2000/1 - 0s - loss: 0.3679 - accuracy: 0.9380
```

Out[39]: [0.2697342637479305, 0.938]

In [40]:
```
history = model1.history.history
plt.title('Model Accuracy')
plt.plot(history['accuracy'], label='Training')
plt.plot(history['val_accuracy'], label='Validation')
plt.xlabel('Epochs')
plt.legend()
plt.show()
```



Optimized Neural Net

In [41]:
```
model2 = Sequential()
model2.add(Dense(50, activation='relu', input_shape=(X_train.shape[1], ),
                 kernel_regularizer=keras.regularizers.l1(.001)))
model2.add(Dropout(rate=.5))
model2.add(Dense(10, activation='relu',
                 kernel_regularizer=keras.regularizers.l1(.001)))
model2.add(Dropout(rate=.5))
model2.add(Dense(10, activation='relu',
                 kernel_regularizer=keras.regularizers.l1(.001)))
model2.add(Dropout(rate=.5))
model2.add(Dense(1, activation='sigmoid'))

model2.compile(optimizer='adam',
               loss='binary_crossentropy',
               metrics=['accuracy'])

early_stopping = EarlyStopping(monitor='val_loss', patience=3)
model2.fit(X_train, y_train, epochs=100, batch_size=50,
          validation_data=(X_valid, y_valid),
          callbacks=[early_stopping], verbose = 2)

model2.evaluate(X_test, y_test, verbose = 2)
```

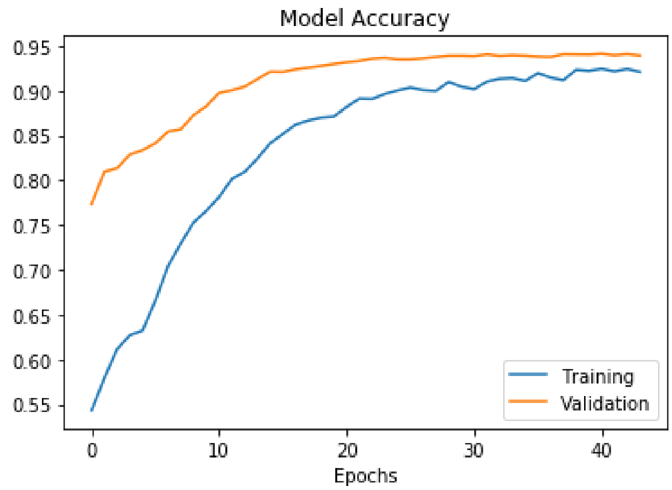
```
Train on 5000 samples, validate on 5000 samples
Epoch 1/100
5000/5000 - 1s - loss: 2.1437 - accuracy: 0.5434 - val_loss: 1.1295 - val_accuracy: 0.77
38
Epoch 2/100
5000/5000 - 1s - loss: 1.4455 - accuracy: 0.5792 - val_loss: 1.1137 - val_accuracy: 0.80
96
Epoch 3/100
5000/5000 - 1s - loss: 1.2802 - accuracy: 0.6116 - val_loss: 1.1001 - val_accuracy: 0.81
36
Epoch 4/100
5000/5000 - 1s - loss: 1.2067 - accuracy: 0.6272 - val_loss: 1.0765 - val_accuracy: 0.82
88
Epoch 5/100
5000/5000 - 1s - loss: 1.1427 - accuracy: 0.6322 - val_loss: 1.0395 - val_accuracy: 0.83
36
Epoch 6/100
5000/5000 - 1s - loss: 1.0933 - accuracy: 0.6656 - val_loss: 0.9850 - val_accuracy: 0.84
14
Epoch 7/100
5000/5000 - 1s - loss: 1.0087 - accuracy: 0.7046 - val_loss: 0.9000 - val_accuracy: 0.85
44
Epoch 8/100
5000/5000 - 1s - loss: 0.9774 - accuracy: 0.7296 - val_loss: 0.8416 - val_accuracy: 0.85
70
Epoch 9/100
5000/5000 - 1s - loss: 0.8988 - accuracy: 0.7530 - val_loss: 0.7652 - val_accuracy: 0.87
28
Epoch 10/100
5000/5000 - 1s - loss: 0.8370 - accuracy: 0.7660 - val_loss: 0.6879 - val_accuracy: 0.88
28
Epoch 11/100
5000/5000 - 1s - loss: 0.7959 - accuracy: 0.7814 - val_loss: 0.6239 - val_accuracy: 0.89
76
Epoch 12/100
5000/5000 - 1s - loss: 0.7397 - accuracy: 0.8016 - val_loss: 0.5688 - val_accuracy: 0.90
06
Epoch 13/100
5000/5000 - 1s - loss: 0.6884 - accuracy: 0.8094 - val_loss: 0.5240 - val_accuracy: 0.90
46
Epoch 14/100
5000/5000 - 1s - loss: 0.6509 - accuracy: 0.8242 - val_loss: 0.4865 - val_accuracy: 0.91
30
Epoch 15/100
5000/5000 - 1s - loss: 0.5949 - accuracy: 0.8412 - val_loss: 0.4485 - val_accuracy: 0.92
12
Epoch 16/100
5000/5000 - 1s - loss: 0.5707 - accuracy: 0.8518 - val_loss: 0.4258 - val_accuracy: 0.92
10
Epoch 17/100
5000/5000 - 1s - loss: 0.5488 - accuracy: 0.8622 - val_loss: 0.4069 - val_accuracy: 0.92
40
Epoch 18/100
5000/5000 - 1s - loss: 0.5295 - accuracy: 0.8668 - val_loss: 0.3939 - val_accuracy: 0.92
58
Epoch 19/100
5000/5000 - 1s - loss: 0.5040 - accuracy: 0.8700 - val_loss: 0.3792 - val_accuracy: 0.92
76
Epoch 20/100
5000/5000 - 1s - loss: 0.4814 - accuracy: 0.8712 - val_loss: 0.3589 - val_accuracy: 0.92
98
Epoch 21/100
5000/5000 - 1s - loss: 0.4584 - accuracy: 0.8820 - val_loss: 0.3507 - val_accuracy: 0.93
```

```
18
Epoch 22/100
5000/5000 - 1s - loss: 0.4437 - accuracy: 0.8912 - val_loss: 0.3424 - val_accuracy: 0.93
32
Epoch 23/100
5000/5000 - 1s - loss: 0.4394 - accuracy: 0.8908 - val_loss: 0.3348 - val_accuracy: 0.93
58
Epoch 24/100
5000/5000 - 1s - loss: 0.4305 - accuracy: 0.8964 - val_loss: 0.3267 - val_accuracy: 0.93
68
Epoch 25/100
5000/5000 - 1s - loss: 0.4144 - accuracy: 0.9004 - val_loss: 0.3206 - val_accuracy: 0.93
50
Epoch 26/100
5000/5000 - 1s - loss: 0.3975 - accuracy: 0.9036 - val_loss: 0.3133 - val_accuracy: 0.93
50
Epoch 27/100
5000/5000 - 1s - loss: 0.3930 - accuracy: 0.9008 - val_loss: 0.3075 - val_accuracy: 0.93
62
Epoch 28/100
5000/5000 - 1s - loss: 0.3968 - accuracy: 0.8998 - val_loss: 0.3042 - val_accuracy: 0.93
76
Epoch 29/100
5000/5000 - 1s - loss: 0.3913 - accuracy: 0.9098 - val_loss: 0.3017 - val_accuracy: 0.93
90
Epoch 30/100
5000/5000 - 1s - loss: 0.3787 - accuracy: 0.9046 - val_loss: 0.2947 - val_accuracy: 0.93
90
Epoch 31/100
5000/5000 - 1s - loss: 0.3857 - accuracy: 0.9018 - val_loss: 0.2936 - val_accuracy: 0.93
86
Epoch 32/100
5000/5000 - 1s - loss: 0.3711 - accuracy: 0.9102 - val_loss: 0.2893 - val_accuracy: 0.94
06
Epoch 33/100
5000/5000 - 0s - loss: 0.3632 - accuracy: 0.9134 - val_loss: 0.2882 - val_accuracy: 0.93
88
Epoch 34/100
5000/5000 - 1s - loss: 0.3682 - accuracy: 0.9142 - val_loss: 0.2902 - val_accuracy: 0.93
96
Epoch 35/100
5000/5000 - 1s - loss: 0.3640 - accuracy: 0.9110 - val_loss: 0.2871 - val_accuracy: 0.93
90
Epoch 36/100
5000/5000 - 1s - loss: 0.3493 - accuracy: 0.9196 - val_loss: 0.2851 - val_accuracy: 0.93
80
Epoch 37/100
5000/5000 - 1s - loss: 0.3648 - accuracy: 0.9148 - val_loss: 0.2860 - val_accuracy: 0.93
76
Epoch 38/100
5000/5000 - 1s - loss: 0.3572 - accuracy: 0.9118 - val_loss: 0.2781 - val_accuracy: 0.94
08
Epoch 39/100
5000/5000 - 1s - loss: 0.3396 - accuracy: 0.9234 - val_loss: 0.2798 - val_accuracy: 0.94
06
Epoch 40/100
5000/5000 - 1s - loss: 0.3363 - accuracy: 0.9222 - val_loss: 0.2722 - val_accuracy: 0.94
04
Epoch 41/100
5000/5000 - 1s - loss: 0.3400 - accuracy: 0.9246 - val_loss: 0.2704 - val_accuracy: 0.94
14
Epoch 42/100
5000/5000 - 1s - loss: 0.3366 - accuracy: 0.9216 - val_loss: 0.2725 - val_accuracy: 0.93
96
```

```
Epoch 43/100
5000/5000 - 1s - loss: 0.3357 - accuracy: 0.9244 - val_loss: 0.2785 - val_accuracy: 0.94
10
Epoch 44/100
5000/5000 - 1s - loss: 0.3304 - accuracy: 0.9210 - val_loss: 0.2770 - val_accuracy: 0.93
92
2000/1 - 0s - loss: 0.2893 - accuracy: 0.9410
```

Out[41]:
```
[0.25987746596336364, 0.941]
```

In [42]:
```
history = model2.history.history
plt.title('Model Accuracy')
plt.plot(history['accuracy'], label='Training')
plt.plot(history['val_accuracy'], label='Validation')
plt.xlabel('Epochs')
plt.legend()
plt.show()
```



Preprocessing data for Convolutional Neural Net

In [30]:
```
X_train = pca.inverse_transform(X_train)
X_test = pca.inverse_transform(X_test)
X_valid = pca.inverse_transform(X_valid)
```

In [31]:
```
X_train = X_train.reshape(-1, 32, 32, 3)
X_valid = X_valid.reshape(-1, 32, 32, 3)
X_test = X_test.reshape(-1, 32, 32, 3)
print('X shapes: ', X_train.shape, X_valid.shape, X_test.shape)
```

```
X shapes:  (5000, 32, 32, 3) (5000, 32, 32, 3) (2000, 32, 32, 3)
```

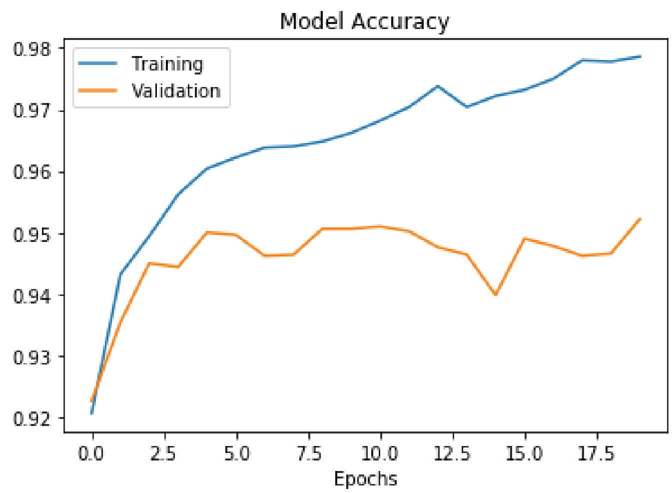
Initial Convolutional Neural Net

In [45]:
```
model3 = Sequential()
model3.add(Conv2D(16, kernel_size=(3, 3), activation='relu',
                 padding='same', input_shape=(32,32,3)))

model3.add(MaxPooling2D(pool_size=(2, 2), strides=2))

model3.add(Flatten())
model3.add(Dense(1, activation='sigmoid'))
model3.compile(optimizer='adam',
               loss='binary_crossentropy',
               metrics=['accuracy'])
model3.fit(X_train, y_train, epochs=20, batch_size=10,
           validation_data=(X_valid, y_valid), verbose = 2)
model3.evaluate(X_test, y_test, verbose = 2)
```

```
Train on 5000 samples, validate on 5000 samples
Epoch 1/20
5000/5000 - 3s - loss: 0.2191 - accuracy: 0.9142 - val_loss: 0.1868 - val_accuracy: 0.92
98
Epoch 2/20
5000/5000 - 3s - loss: 0.1620 - accuracy: 0.9422 - val_loss: 0.1784 - val_accuracy: 0.93
42
Epoch 3/20
5000/5000 - 3s - loss: 0.1427 - accuracy: 0.9502 - val_loss: 0.1530 - val_accuracy: 0.94
24
Epoch 4/20
5000/5000 - 3s - loss: 0.1263 - accuracy: 0.9534 - val_loss: 0.1652 - val_accuracy: 0.94
04
Epoch 5/20
5000/5000 - 3s - loss: 0.1147 - accuracy: 0.9600 - val_loss: 0.1513 - val_accuracy: 0.94
32
Epoch 6/20
5000/5000 - 3s - loss: 0.1091 - accuracy: 0.9612 - val_loss: 0.1441 - val_accuracy: 0.94
84
Epoch 7/20
5000/5000 - 3s - loss: 0.1044 - accuracy: 0.9600 - val_loss: 0.1425 - val_accuracy: 0.94
74
Epoch 8/20
5000/5000 - 3s - loss: 0.0981 - accuracy: 0.9646 - val_loss: 0.1517 - val_accuracy: 0.94
28
Epoch 9/20
5000/5000 - 3s - loss: 0.0941 - accuracy: 0.9660 - val_loss: 0.1427 - val_accuracy: 0.95
28
Epoch 10/20
5000/5000 - 3s - loss: 0.0884 - accuracy: 0.9684 - val_loss: 0.1471 - val_accuracy: 0.95
34
Epoch 11/20
5000/5000 - 3s - loss: 0.0862 - accuracy: 0.9680 - val_loss: 0.1641 - val_accuracy: 0.94
46
Epoch 12/20
5000/5000 - 3s - loss: 0.0895 - accuracy: 0.9670 - val_loss: 0.1707 - val_accuracy: 0.94
56
Epoch 13/20
5000/5000 - 3s - loss: 0.0792 - accuracy: 0.9724 - val_loss: 0.1545 - val_accuracy: 0.94
94
Epoch 14/20
5000/5000 - 3s - loss: 0.0780 - accuracy: 0.9716 - val_loss: 0.1893 - val_accuracy: 0.94
16
Epoch 15/20
5000/5000 - 3s - loss: 0.0742 - accuracy: 0.9722 - val_loss: 0.1516 - val_accuracy: 0.94
80
Epoch 16/20
5000/5000 - 3s - loss: 0.0718 - accuracy: 0.9730 - val_loss: 0.1447 - val_accuracy: 0.94
98
Epoch 17/20
5000/5000 - 3s - loss: 0.0705 - accuracy: 0.9734 - val_loss: 0.1559 - val_accuracy: 0.94
74
Epoch 18/20
5000/5000 - 3s - loss: 0.0656 - accuracy: 0.9736 - val_loss: 0.1515 - val_accuracy: 0.95
10
Epoch 19/20
5000/5000 - 3s - loss: 0.0666 - accuracy: 0.9758 - val_loss: 0.1714 - val_accuracy: 0.94
82
Epoch 20/20
5000/5000 - 3s - loss: 0.0620 - accuracy: 0.9756 - val_loss: 0.1638 - val_accuracy: 0.94
90
2000/1 - 0s - loss: 0.3408 - accuracy: 0.9560
```

Out[45]: [0.1358242617174983, 0.956]

In [30]:
```
history = model3.history.history
plt.title('Model Accuracy')
plt.plot(history['accuracy'], label='Training')
plt.plot(history['val_accuracy'], label='Validation')
plt.xlabel('Epochs')
plt.legend()
plt.show()
```



Optimized Convolutional Neural Net

In [34]:

```
model4 = Sequential()
model4.add(Conv2D(16, kernel_size=3, activation='relu', strides=1,
                 padding='same', input_shape=(32,32,3),
                 kernel_regularizer=keras.regularizers.l2(.001)))
model4.add(Dropout(rate=.5))
model4.add(MaxPooling2D(pool_size=4, strides=2))

model4.add(Conv2D(32, kernel_size=3, activation='relu',
                 padding='same',
                 kernel_regularizer=keras.regularizers.l2(.001)))
model4.add(Dropout(rate=.5))
model4.add(MaxPooling2D(pool_size=4, strides=3))

model4.add(Flatten())
model4.add(Dense(1, activation='sigmoid'))

model4.compile(optimizer='adam',
               loss='binary_crossentropy',
               metrics=['accuracy'])
early_stopping = EarlyStopping(monitor='val_loss', patience=3)
model4.fit(X_train, y_train, epochs=200, batch_size=50,
           validation_data=(X_valid, y_valid), verbose = 2, callbacks=[early_stopping])
model4.evaluate(X_test, y_test, verbose = 2)
```

```
Train on 5000 samples, validate on 5000 samples
Epoch 1/200
5000/5000 - 2s - loss: 0.2944 - accuracy: 0.8900 - val_loss: 0.4036 - val_accuracy: 0.92
20
Epoch 2/200
5000/5000 - 1s - loss: 0.2112 - accuracy: 0.9306 - val_loss: 0.3595 - val_accuracy: 0.93
38
Epoch 3/200
5000/5000 - 1s - loss: 0.1854 - accuracy: 0.9404 - val_loss: 0.3470 - val_accuracy: 0.93
60
Epoch 4/200
5000/5000 - 1s - loss: 0.1746 - accuracy: 0.9444 - val_loss: 0.3248 - val_accuracy: 0.94
10
Epoch 5/200
5000/5000 - 1s - loss: 0.1650 - accuracy: 0.9482 - val_loss: 0.3215 - val_accuracy: 0.94
44
Epoch 6/200
5000/5000 - 1s - loss: 0.1527 - accuracy: 0.9498 - val_loss: 0.3142 - val_accuracy: 0.94
22
Epoch 7/200
5000/5000 - 1s - loss: 0.1441 - accuracy: 0.9544 - val_loss: 0.2990 - val_accuracy: 0.95
36
Epoch 8/200
5000/5000 - 1s - loss: 0.1437 - accuracy: 0.9542 - val_loss: 0.2972 - val_accuracy: 0.94
96
Epoch 9/200
5000/5000 - 1s - loss: 0.1345 - accuracy: 0.9604 - val_loss: 0.2983 - val_accuracy: 0.94
30
Epoch 10/200
5000/5000 - 1s - loss: 0.1296 - accuracy: 0.9624 - val_loss: 0.2805 - val_accuracy: 0.95
14
Epoch 11/200
5000/5000 - 1s - loss: 0.1279 - accuracy: 0.9612 - val_loss: 0.2818 - val_accuracy: 0.95
70
Epoch 12/200
5000/5000 - 1s - loss: 0.1233 - accuracy: 0.9632 - val_loss: 0.2623 - val_accuracy: 0.95
30
Epoch 13/200
5000/5000 - 1s - loss: 0.1167 - accuracy: 0.9682 - val_loss: 0.2558 - val_accuracy: 0.95
72
Epoch 14/200
5000/5000 - 1s - loss: 0.1189 - accuracy: 0.9628 - val_loss: 0.2656 - val_accuracy: 0.94
72
Epoch 15/200
5000/5000 - 1s - loss: 0.1250 - accuracy: 0.9614 - val_loss: 0.2575 - val_accuracy: 0.95
16
Epoch 16/200
5000/5000 - 1s - loss: 0.1099 - accuracy: 0.9674 - val_loss: 0.2528 - val_accuracy: 0.95
36
Epoch 17/200
5000/5000 - 1s - loss: 0.1080 - accuracy: 0.9720 - val_loss: 0.2577 - val_accuracy: 0.95
88
Epoch 18/200
5000/5000 - 1s - loss: 0.1114 - accuracy: 0.9686 - val_loss: 0.2579 - val_accuracy: 0.96
16
Epoch 19/200
5000/5000 - 1s - loss: 0.0997 - accuracy: 0.9706 - val_loss: 0.2506 - val_accuracy: 0.95
04
Epoch 20/200
5000/5000 - 1s - loss: 0.1034 - accuracy: 0.9702 - val_loss: 0.2393 - val_accuracy: 0.95
16
Epoch 21/200
5000/5000 - 1s - loss: 0.1005 - accuracy: 0.9716 - val_loss: 0.2280 - val_accuracy: 0.96
```

```
24
Epoch 22/200
5000/5000 - 1s - loss: 0.0971 - accuracy: 0.9726 - val_loss: 0.2268 - val_accuracy: 0.95
92
Epoch 23/200
5000/5000 - 1s - loss: 0.1047 - accuracy: 0.9696 - val_loss: 0.2301 - val_accuracy: 0.95
94
Epoch 24/200
5000/5000 - 1s - loss: 0.0928 - accuracy: 0.9746 - val_loss: 0.2225 - val_accuracy: 0.95
98
Epoch 25/200
5000/5000 - 1s - loss: 0.0934 - accuracy: 0.9736 - val_loss: 0.2296 - val_accuracy: 0.95
66
Epoch 26/200
5000/5000 - 1s - loss: 0.0922 - accuracy: 0.9740 - val_loss: 0.2254 - val_accuracy: 0.95
52
Epoch 27/200
5000/5000 - 1s - loss: 0.0967 - accuracy: 0.9720 - val_loss: 0.2240 - val_accuracy: 0.96
06
2000/1 - 0s - loss: 0.2431 - accuracy: 0.9650
```

Out[34]: [0.22146977257728576, 0.965]

In [35]:

```
history = model4.history.history
plt.title('Model Accuracy')
plt.plot(history['accuracy'], label='Training')
plt.plot(history['val_accuracy'], label='Validation')
plt.xlabel('Epochs')
plt.legend()
plt.show()
```