# STAT 542: Midterm Project

## Face Recognition


- Yu-Ching Liao ycliao3@illinois.edu


# Basic Import

## Package Import

In [1]:
```
# import necessary libraries and modules
from sklearn.datasets import fetch_olivetti_faces
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from keras.wrappers.scikit_learn import KerasClassifier
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
import time
```

## Dataset Import

In [39]:
```
# load the Olivetti faces dataset
data = fetch_olivetti_faces()
X = data['data']
y = data['target']
```

## Prepare train and test set

In [40]:
```python
# split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar
```

# Statistical Learning

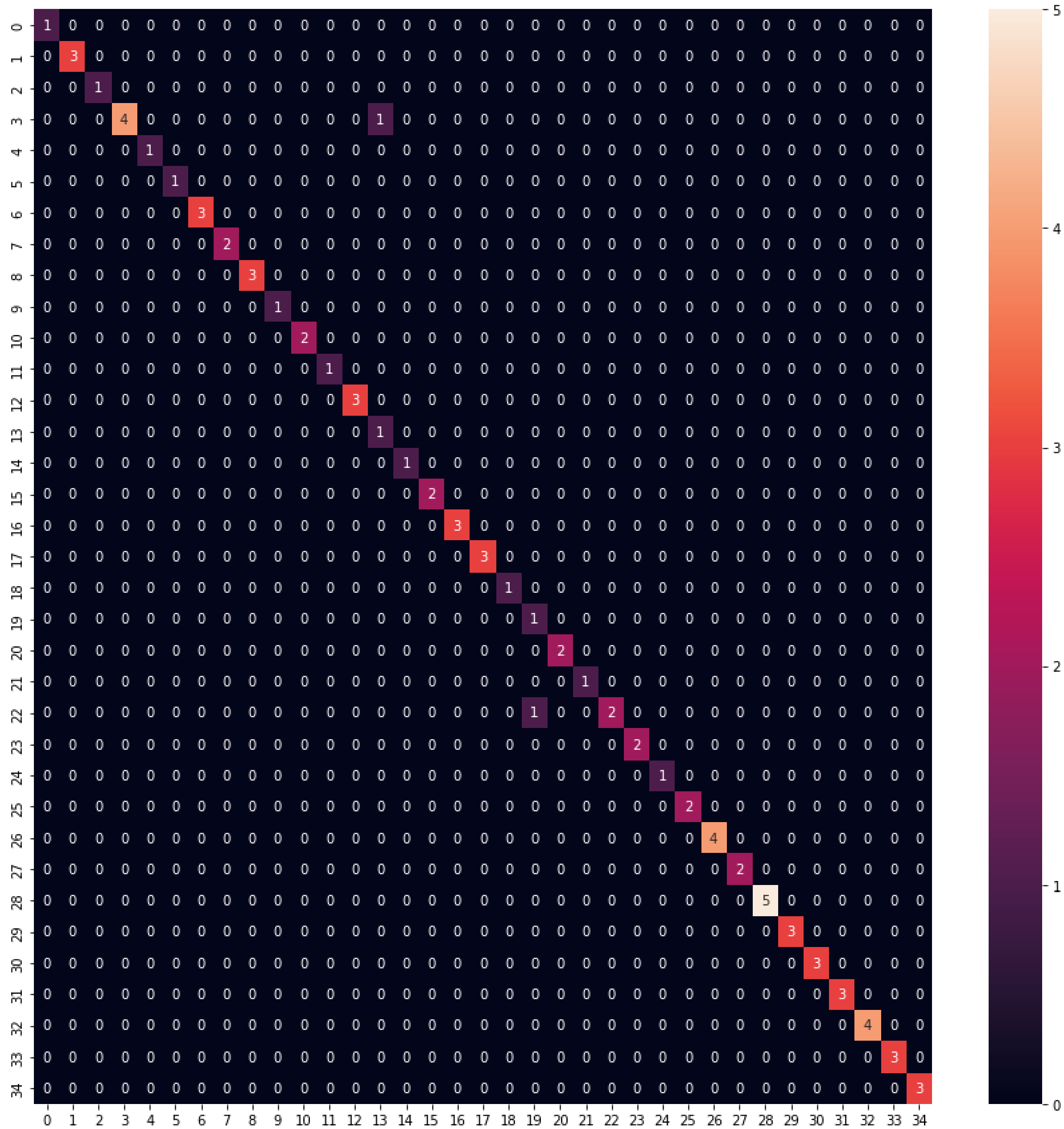
## Running the Model without PCA

### Logistic

In [6]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
import seaborn as sns

start = time.time()
# Create a logistic regression model
clf = LogisticRegression()

# Fit the model to the training data
clf.fit(X_train, y_train)

# Use the model to make predictions on the testing data
y_pred = clf.predict(X_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
end = time.time()

print("Out-sample Accuracy for logistic regression:", accuracy)
print("Time Comsumption:", end - start, "sec.")

cm = confusion_matrix(y_test, y_pred)

fig, ax = plt.subplots(figsize=(15,15))
sns.heatmap(cm, annot=True, fmt='d', ax=ax)
```

```
Accuracy for logistic regression: 0.975
Time Comsumption: 6.632861137390137 sec.
```

Out[6]:
```
<Axes: >
```



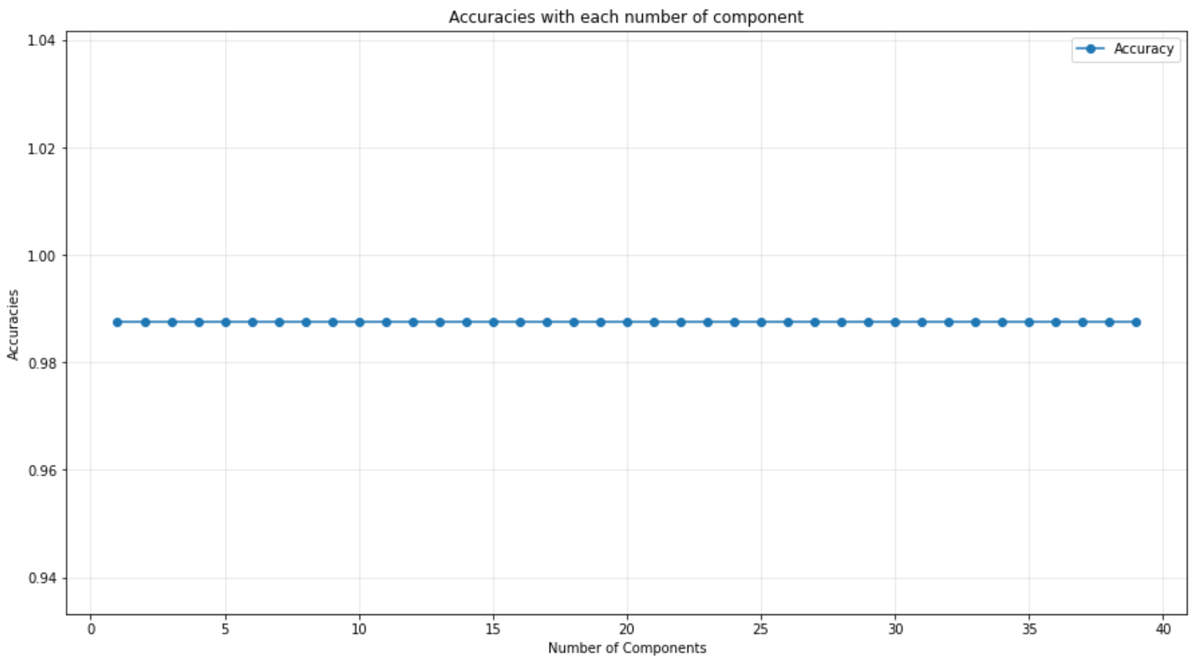
## LDA

In [55]:

```
#find the best parameters
acc = []
for n in range(1, 40):
  lda = LinearDiscriminantAnalysis(n_components=n)
  lda.fit(X_train, y_train)
  y_pred = lda.predict(X_test)
  accuracy = accuracy_score(y_test, y_pred)
  if n % 5 == 0:
    print("Out-sample Accuracy for LDA for", n, "components:", accuracy)

  acc.append(accuracy)
  accuracy = 0
plt.figure(figsize=(15,8))
plt.title("Accuracies with each number of component")
```

```
plt.plot(range(1,40), acc, 'o-',  label = "Accuracy")
plt.ylabel("Accuracies")
plt.xlabel("Number of Components")
plt.legend()
plt.grid(alpha=0.3)
plt.show()

print("The best number of components:", max(acc))
```

```
Out-sample Accuracy for LDA for 5 components: 0.9875
Out-sample Accuracy for LDA for 10 components: 0.9875
Out-sample Accuracy for LDA for 15 components: 0.9875
Out-sample Accuracy for LDA for 20 components: 0.9875
Out-sample Accuracy for LDA for 25 components: 0.9875
Out-sample Accuracy for LDA for 30 components: 0.9875
Out-sample Accuracy for LDA for 35 components: 0.9875
```




```
The best number of components: 0.9875
```

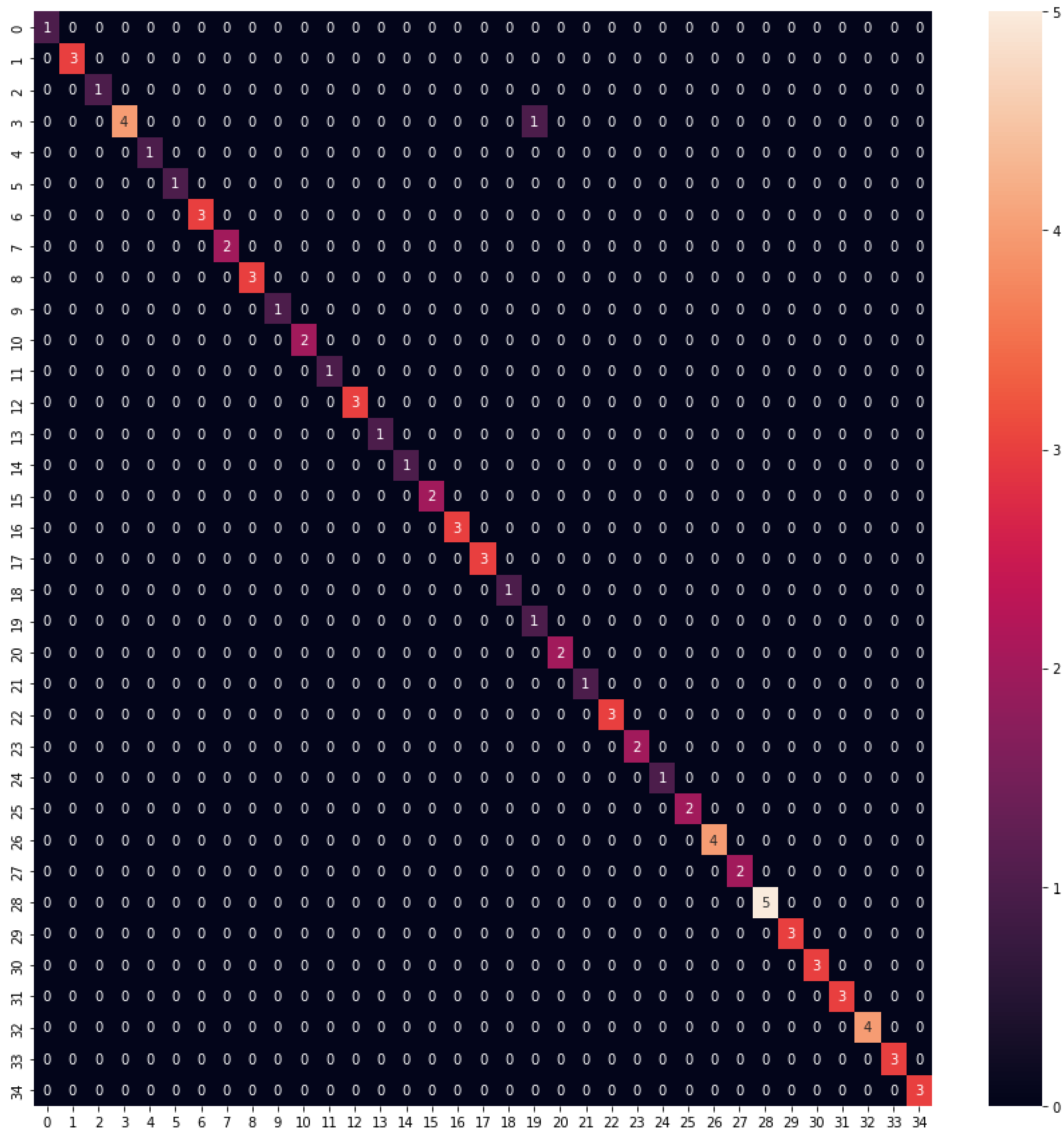
So basically all of those number of components have same performance so we are going to randomly pick a number ranged from 1 to 39 and see its performance.

In [8]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import accuracy_score
from mlxtend.plotting import plot_decision_regions

start = time.time()
# Create an instance of LDA
lda = LinearDiscriminantAnalysis(n_components=39)

# Train the LDA model
lda.fit(X_train, y_train)


# Make predictions on the test set
y_pred = lda.predict(X_test)
```

```python
# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

end = time.time()

# Print the accuracy
print("Out-sample Accuracy for LDA:", accuracy)
print("Time Comsumption:", end - start, "sec.")

cm = confusion_matrix(y_test, y_pred)

fig, ax = plt.subplots(figsize=(15,15))
sns.heatmap(cm, annot=True, fmt='d', ax=ax)
```

```
Out-sample Accuracy for LDA: 0.9875
Time Comsumption: 0.794893741607666 sec.
```

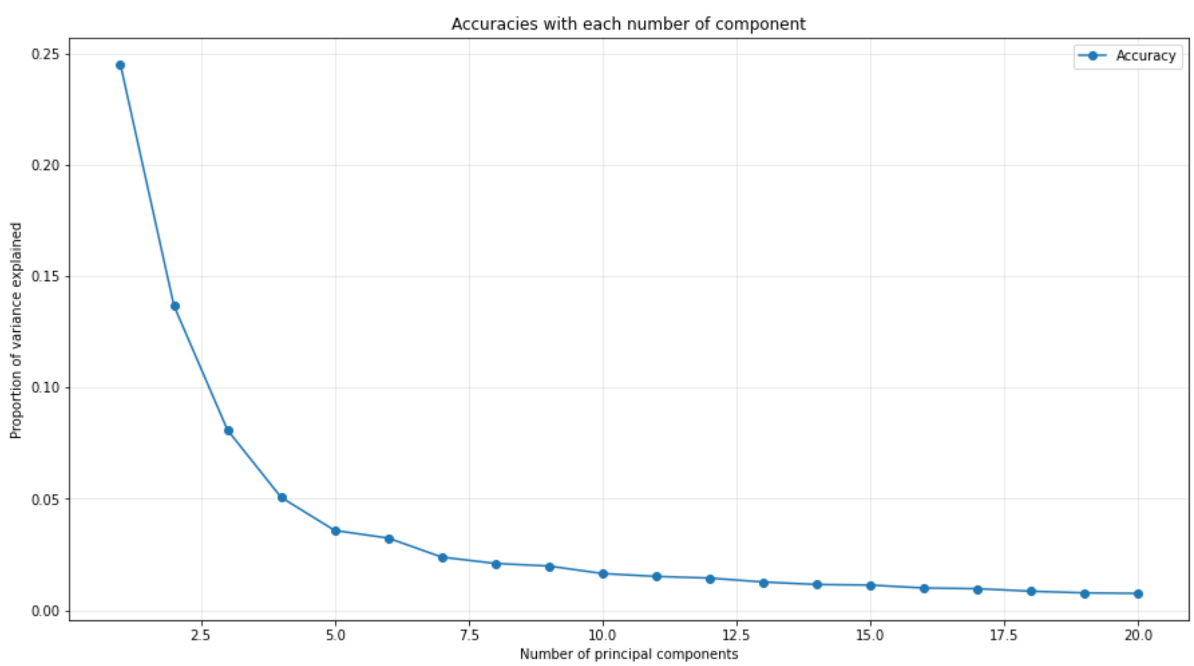
Out[8]: <Axes: >



# Running the Model with PCA

# PCA Visualization

In [56]:
```
from sklearn.decomposition import PCA

# load the Olivetti faces dataset
data = fetch_olivetti_faces()
X = data['data']
y = data['target']

# split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran

pca = PCA(n_components=20)
X_train_pca = pca.fit_transform(X_train)

# Apply PCA to the testing set
X_test_pca = pca.transform(X_test)

# visualizw results
plt.figure(figsize=(15,8))
plt.plot(range(1, pca.n_components_+1), pca.explained_variance_ratio_, 'o-',
plt.title("Accuracies with each number of component")
plt.xlabel('Number of principal components')
plt.ylabel('Proportion of variance explained')
plt.legend()
plt.grid(alpha=0.3)
plt.show()
```

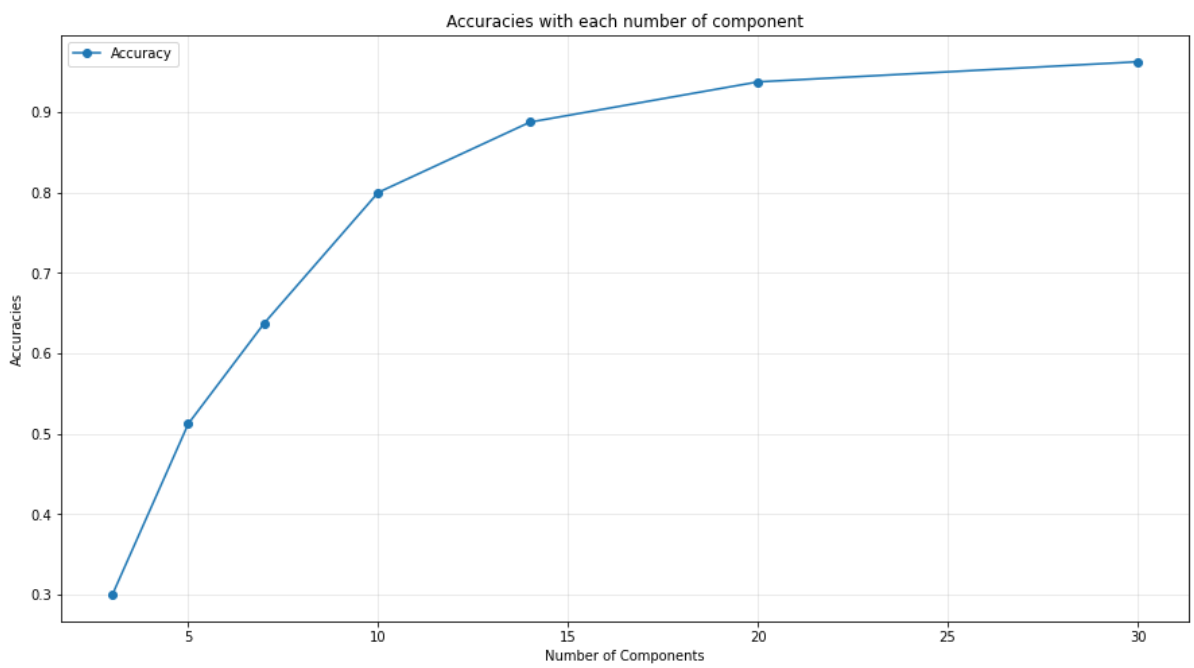


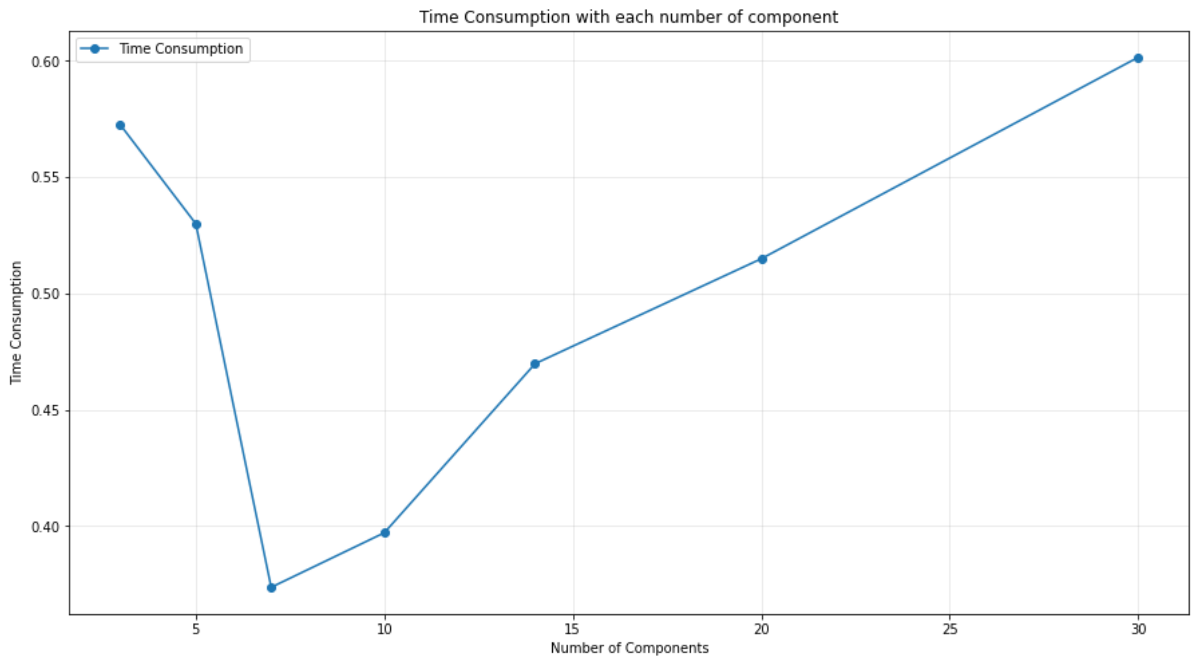

We are going to try the number of components with 3, 5, 7, 10, 14, 20, 30 to see the performance of PCA.

## Logistic with PCA

In [62]:
```
acc = []
time_comp = []
for n in [3, 5, 7, 10, 14, 20, 30]:
  start = time.time()
  pca = PCA(n_components=n)
  X_train_pca = pca.fit_transform(X_train)
  X_test_pca = pca.transform(X_test)
  clf = LogisticRegression()
  clf.fit(X_train_pca, y_train)
  y_pred = clf.predict(X_test_pca)
  accuracy = accuracy_score(y_test, y_pred)
  end = time.time()
  print("Number of components:", n, ", Out-sample Accuracy for logistic regr
  print("Time Comsumption:", end - start, "sec.\n")
  acc.append(accuracy)
  time_comp.append(end - start)

plt.figure(figsize=(15,8))
plt.plot([3, 5, 7, 10, 14, 20, 30], acc, 'o-', label = 'Accuracy')
plt.ylabel("Accuracies")
plt.xlabel("Number of Components")
plt.title("Accuracies with each number of component")
plt.legend()
plt.grid(alpha=0.3)
plt.show()

plt.figure(figsize=(15,8))
plt.plot([3, 5, 7, 10, 14, 20, 30], time_comp, 'o-', label = 'Time Consumpti
plt.ylabel("Time Consumption")
plt.xlabel("Number of Components")
plt.title("Time Consumption with each number of component")
plt.legend()
plt.grid(alpha=0.3)
plt.show()
```

```
Number of components: 3 , Out-sample Accuracy for logistic regression with
PCA: 0.3
Time Comsumption: 0.5724036693572998 sec.

Number of components: 5 , Out-sample Accuracy for logistic regression with
PCA: 0.5125
Time Comsumption: 0.5299580097198486 sec.

Number of components: 7 , Out-sample Accuracy for logistic regression with
PCA: 0.6375
Time Comsumption: 0.3737456798553467 sec.

Number of components: 10 , Out-sample Accuracy for logistic regression with
PCA: 0.8
Time Comsumption: 0.39716148376464844 sec.

Number of components: 14 , Out-sample Accuracy for logistic regression with
PCA: 0.8875
Time Comsumption: 0.46981143951416016 sec.

Number of components: 20 , Out-sample Accuracy for logistic regression with
PCA: 0.9375
Time Comsumption: 0.514904260635376 sec.

Number of components: 30 , Out-sample Accuracy for logistic regression with
PCA: 0.9625
Time Comsumption: 0.6014773845672607 sec.
```



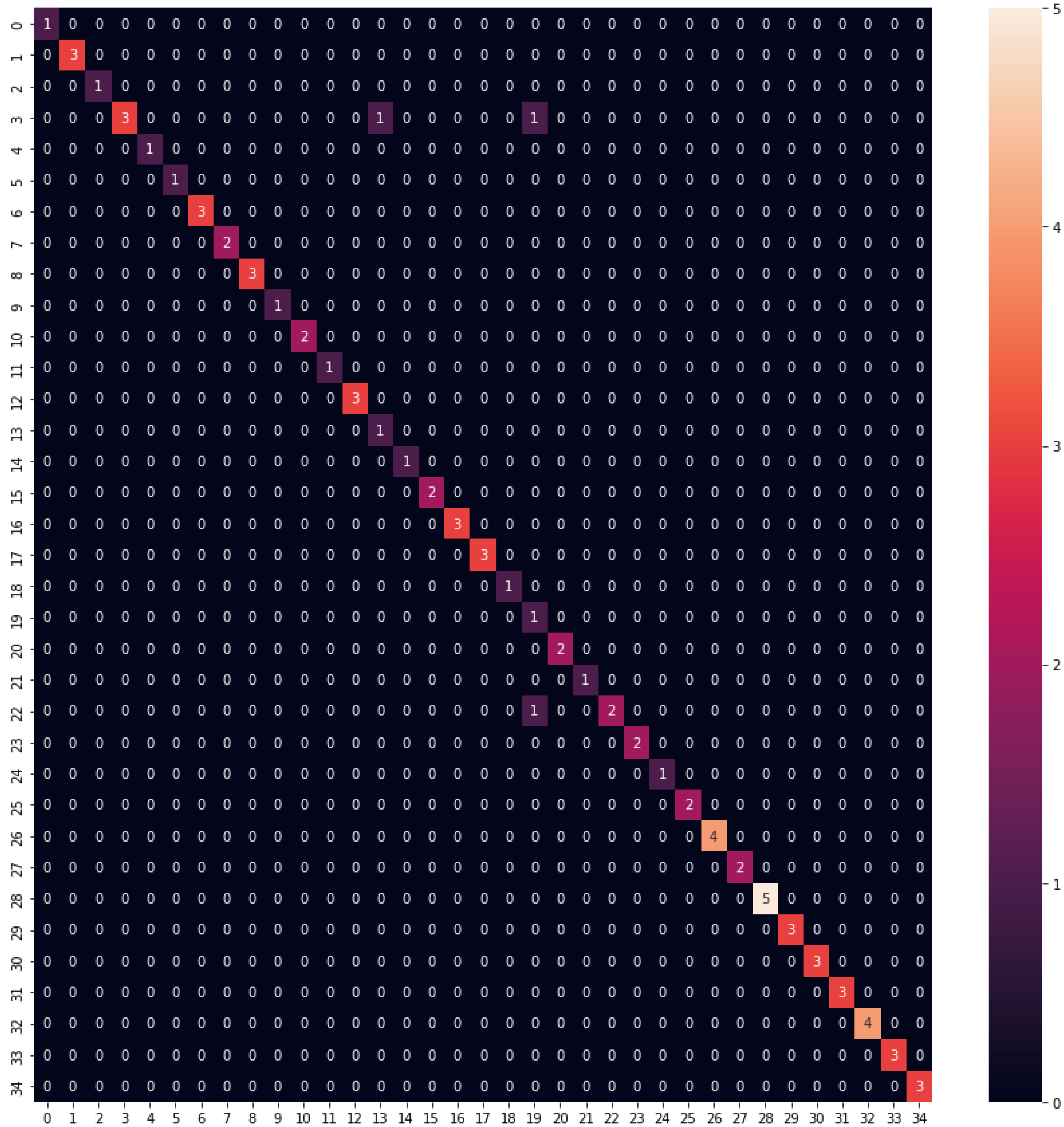



It seemed like the differences among time consumptions are not significant so we will try the number of conponents that have largest accuracy.

In [67]:
```
start = time.time()

#Implementing PCA
pca = PCA(n_components=30)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

# Create a logistic regression model
clf = LogisticRegression()

# Fit the model to the training data
clf.fit(X_train_pca, y_train)

# Use the model to make predictions on the testing data
y_pred = clf.predict(X_test_pca)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
end = time.time()

print("Number of components: 30")
print("Out-sample Accuracy for logistic regression with PCA:", accuracy)
print("Time Comsumption:", end - start, "sec.")

cm = confusion_matrix(y_test, y_pred)

fig, ax = plt.subplots(figsize=(15,15))
sns.heatmap(cm, annot=True, fmt='d', ax=ax)
```

```
Number of components: 30
Out-sample Accuracy for logistic regression with PCA: 0.9625
Time Comsumption: 0.4608933925628662 sec.
```

Out[67]: <Axes: >

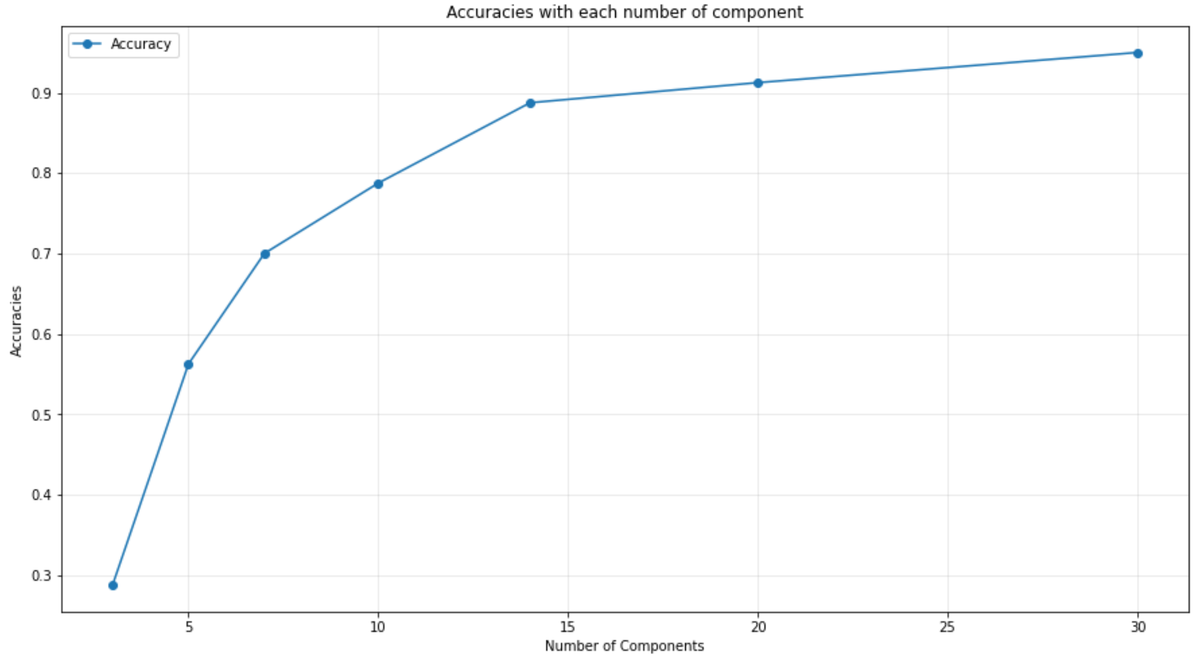


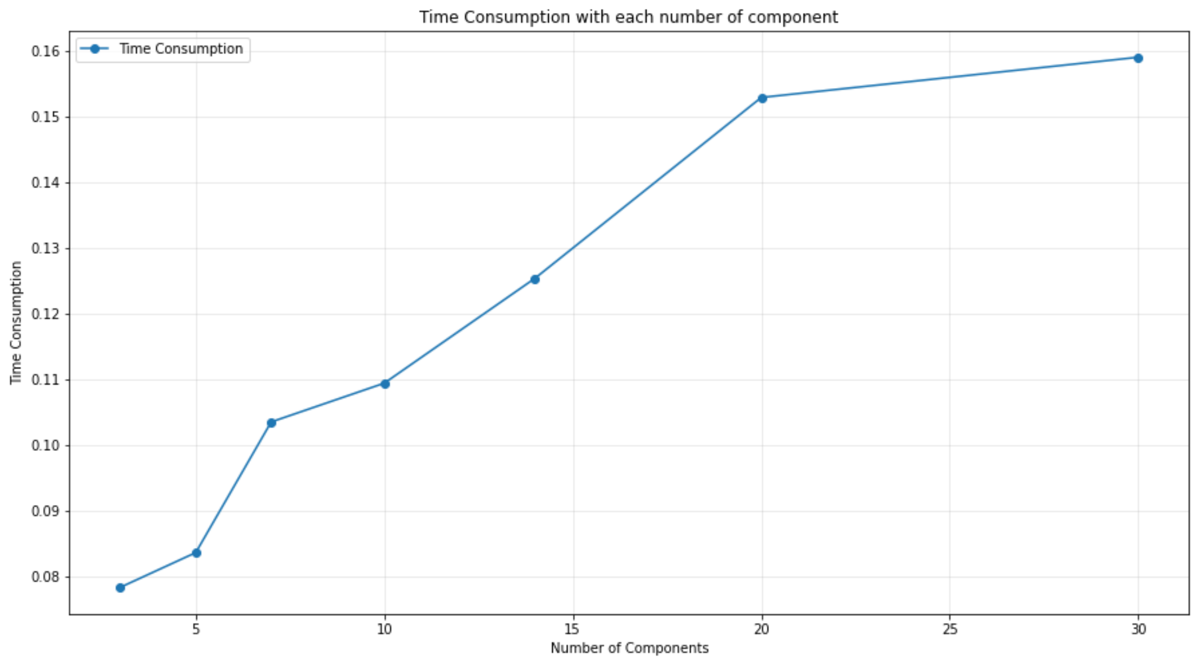
## LDA with PCA

In [64]:
```
acc = []
time_comp = []
for n in [3, 5, 7, 10, 14, 20, 30]:
  start = time.time()
  pca = PCA(n_components=n)
  X_train_pca = pca.fit_transform(X_train)
  X_test_pca = pca.transform(X_test)
  lda = LinearDiscriminantAnalysis(n_components=n-2)
  lda.fit(X_train_pca, y_train)
  y_pred = lda.predict(X_test_pca)
  accuracy = accuracy_score(y_test, y_pred)
  end = time.time()
  print("Number of components:", n, ", Out-sample Accuracy for LDA with PCA:
```

```python
    print("Time Comsumption:", end - start, "sec.\n")
    acc.append(accuracy)
    time_comp.append(end - start)

plt.figure(figsize=(15,8))
plt.plot([3, 5, 7, 10, 14, 20, 30], acc, 'o-', label = 'Accuracy')
plt.ylabel("Accuracies")
plt.xlabel("Number of Components")
plt.title("Accuracies with each number of component")
plt.legend()
plt.grid(alpha=0.3)
plt.show()

plt.figure(figsize=(15,8))
plt.plot([3, 5, 7, 10, 14, 20, 30], time_comp, 'o-', label = 'Time Consumpti
plt.ylabel("Time Consumption")
plt.xlabel("Number of Components")
plt.title("Time Consumption with each number of component")
plt.legend()
plt.grid(alpha=0.3)
plt.show()
```

```
Number of components: 3 , Out-sample Accuracy for LDA with PCA: 0.2875
Time Comsumption: 0.07838582992553711 sec.

Number of components: 5 , Out-sample Accuracy for LDA with PCA: 0.5625
Time Comsumption: 0.08365988731384277 sec.

Number of components: 7 , Out-sample Accuracy for LDA with PCA: 0.7
Time Comsumption: 0.10351896286010742 sec.

Number of components: 10 , Out-sample Accuracy for LDA with PCA: 0.7875
Time Comsumption: 0.10941696166992188 sec.

Number of components: 14 , Out-sample Accuracy for LDA with PCA: 0.8875
Time Comsumption: 0.12537312507629395 sec.

Number of components: 20 , Out-sample Accuracy for LDA with PCA: 0.9125
Time Comsumption: 0.1528491973876953 sec.

Number of components: 30 , Out-sample Accuracy for LDA with PCA: 0.95
Time Comsumption: 0.1589670181274414 sec.
```



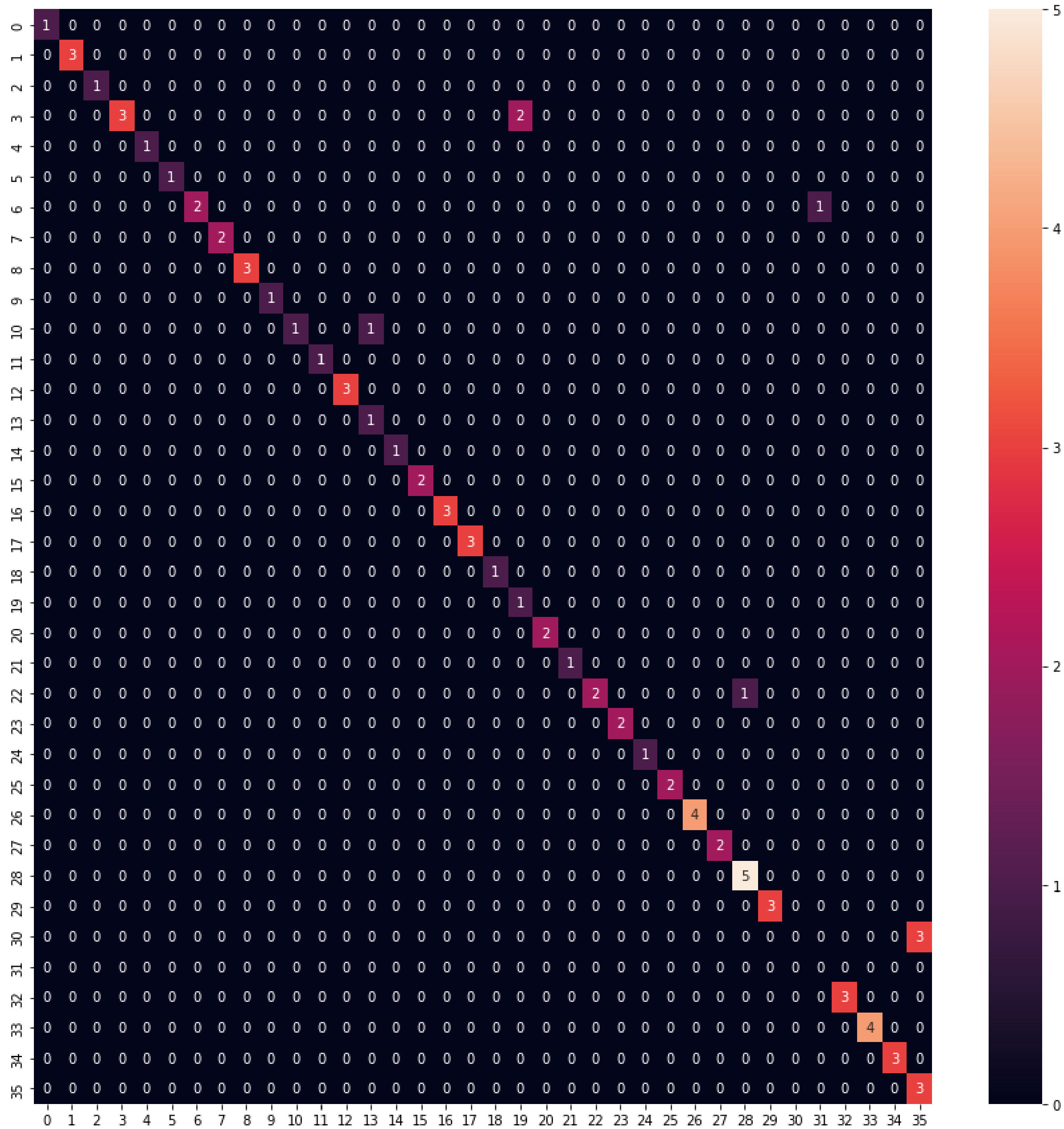

The time consumption will increase nearly twice as the number of components grown from 3 to 30. However, it seemed like the accuracies cease growing since the number of components reached 15. Thus we try number of components with 15 following.

In [66]:
```
start = time.time()

#PCA
pca = PCA(n_components=15)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

# Create an instance of LDA
lda = LinearDiscriminantAnalysis(n_components=5)

# Train the LDA model
```

```python
lda.fit(X_train_pca, y_train)


# Make predictions on the test set
y_pred = lda.predict(X_test_pca)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

end = time.time()

# Print the accuracy
print("Number of conponents: 15")
print("Out-sample Accuracy for LDA:", accuracy)
print("Time Comsumption:", end - start, "sec.")

cm = confusion_matrix(y_test, y_pred)

fig, ax = plt.subplots(figsize=(15,15))
sns.heatmap(cm, annot=True, fmt='d', ax=ax)
```

```
Number of conponents: 15
Out-sample Accuracy for LDA: 0.9
Time Comsumption: 0.28126072883605957 sec.
```

Out[66]: <Axes: >



In [ ]:

# Deep Learning: CNN

In [14]:

```
# Import the necessary libraries
import numpy as np
from sklearn.datasets import fetch_olivetti_faces
from sklearn.model_selection import train_test_split
from tensorflow.keras import layers, models

# Load the Olivetti faces dataset
dataset = fetch_olivetti_faces()
X = dataset.data.reshape(-1, 64, 64, 1)  # Reshape the data into images
y = dataset.target
```

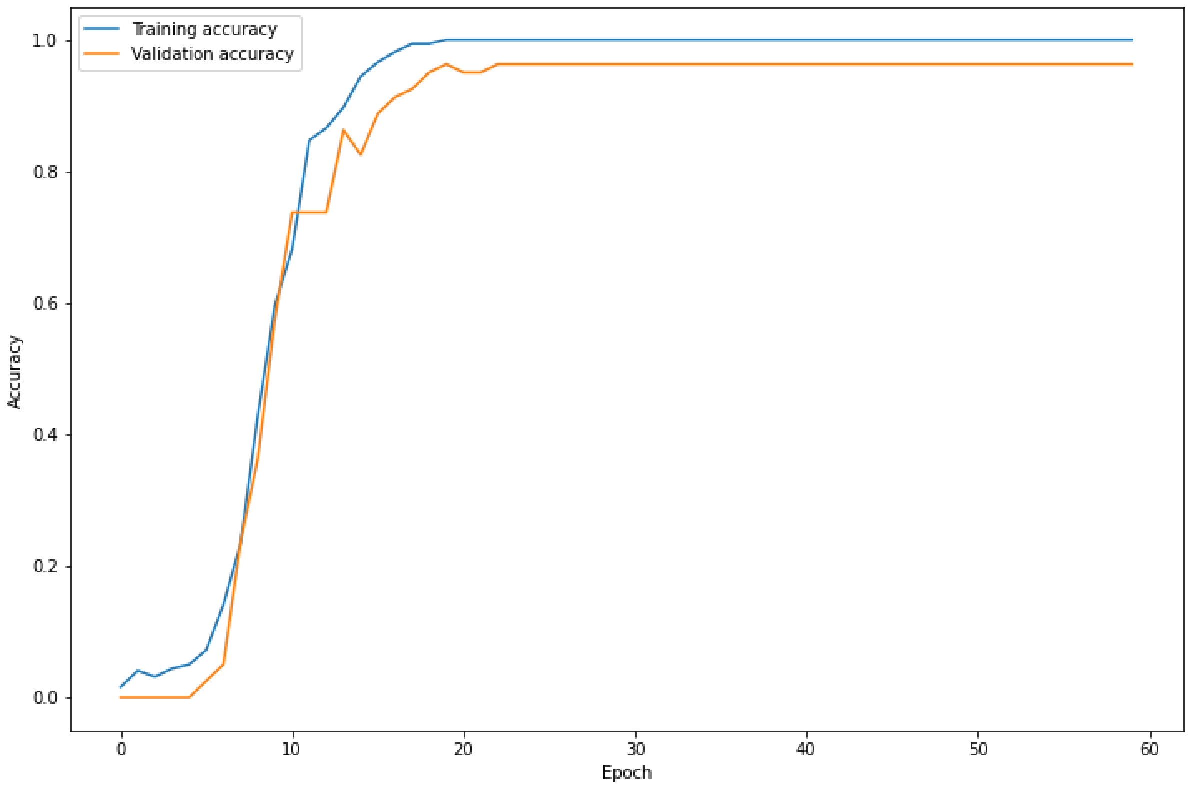
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran

start = time.time()
# Define the CNN model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 1)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(40, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=60, batch_size=32, validation_d


# Evaluate the model on the test data
test_loss, test_acc = model.evaluate(X_test, y_test)

end = time.time()
print('Out-sample Test accuracy for CNN:', test_acc)
print("Time Comsumption:", end - start, "sec.")

# Plot the training and validation accuracy history
plt.figure(figsize=(12,8))
plt.plot(history.history['accuracy'], label='Training accuracy')
plt.plot(history.history['val_accuracy'], label='Validation accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
Epoch 1/60
10/10 [==============================] - 3s 228ms/step - loss: 3.7012 - acc
uracy: 0.0156 - val_loss: 3.6956 - val_accuracy: 0.0000e+00
Epoch 2/60
10/10 [==============================] - 2s 206ms/step - loss: 3.6867 - acc
uracy: 0.0406 - val_loss: 3.7047 - val_accuracy: 0.0000e+00
Epoch 3/60
10/10 [==============================] - 2s 204ms/step - loss: 3.6826 - acc
uracy: 0.0312 - val_loss: 3.7319 - val_accuracy: 0.0000e+00
Epoch 4/60
10/10 [==============================] - 3s 328ms/step - loss: 3.6654 - acc
uracy: 0.0437 - val_loss: 3.7314 - val_accuracy: 0.0000e+00
Epoch 5/60
10/10 [==============================] - 2s 212ms/step - loss: 3.6479 - acc
uracy: 0.0500 - val_loss: 3.7185 - val_accuracy: 0.0000e+00
Epoch 6/60
10/10 [==============================] - 2s 210ms/step - loss: 3.5900 - acc
uracy: 0.0719 - val_loss: 3.6808 - val_accuracy: 0.0250
Epoch 7/60
10/10 [==============================] - 2s 209ms/step - loss: 3.3892 - acc
uracy: 0.1406 - val_loss: 3.5236 - val_accuracy: 0.0500
Epoch 8/60
10/10 [==============================] - 2s 210ms/step - loss: 2.9935 - acc
uracy: 0.2344 - val_loss: 3.0909 - val_accuracy: 0.2375
Epoch 9/60
10/10 [==============================] - 2s 222ms/step - loss: 2.2842 - acc
uracy: 0.4281 - val_loss: 2.5594 - val_accuracy: 0.3625
Epoch 10/60
10/10 [==============================] - 4s 423ms/step - loss: 1.5825 - acc
uracy: 0.5969 - val_loss: 1.6420 - val_accuracy: 0.5750
Epoch 11/60
10/10 [==============================] - 4s 370ms/step - loss: 1.1012 - acc
uracy: 0.6812 - val_loss: 1.0202 - val_accuracy: 0.7375
Epoch 12/60
10/10 [==============================] - 4s 379ms/step - loss: 0.6678 - acc
uracy: 0.8469 - val_loss: 1.0121 - val_accuracy: 0.7375
Epoch 13/60
10/10 [==============================] - 4s 420ms/step - loss: 0.4729 - acc
uracy: 0.8656 - val_loss: 0.9264 - val_accuracy: 0.7375
Epoch 14/60
10/10 [==============================] - 4s 355ms/step - loss: 0.3435 - acc
uracy: 0.8969 - val_loss: 0.4453 - val_accuracy: 0.8625
Epoch 15/60
10/10 [==============================] - 2s 225ms/step - loss: 0.2453 - acc
uracy: 0.9438 - val_loss: 0.4538 - val_accuracy: 0.8250
Epoch 16/60
10/10 [==============================] - 2s 211ms/step - loss: 0.1426 - acc
uracy: 0.9656 - val_loss: 0.4761 - val_accuracy: 0.8875
Epoch 17/60
10/10 [==============================] - 2s 215ms/step - loss: 0.1031 - acc
uracy: 0.9812 - val_loss: 0.3277 - val_accuracy: 0.9125
Epoch 18/60
10/10 [==============================] - 3s 337ms/step - loss: 0.0666 - acc
uracy: 0.9937 - val_loss: 0.3167 - val_accuracy: 0.9250
Epoch 19/60
10/10 [==============================] - 2s 207ms/step - loss: 0.0471 - acc
```

```
uracy: 0.9937 - val_loss: 0.2262 - val_accuracy: 0.9500
Epoch 20/60
10/10 [==============================] - 2s 213ms/step - loss: 0.0283 - acc
uracy: 1.0000 - val_loss: 0.1460 - val_accuracy: 0.9625
Epoch 21/60
10/10 [==============================] - 2s 222ms/step - loss: 0.0189 - acc
uracy: 1.0000 - val_loss: 0.2014 - val_accuracy: 0.9500
Epoch 22/60
10/10 [==============================] - 2s 212ms/step - loss: 0.0112 - acc
uracy: 1.0000 - val_loss: 0.1818 - val_accuracy: 0.9500
Epoch 23/60
10/10 [==============================] - 3s 295ms/step - loss: 0.0091 - acc
uracy: 1.0000 - val_loss: 0.1586 - val_accuracy: 0.9625
Epoch 24/60
10/10 [==============================] - 3s 246ms/step - loss: 0.0072 - acc
uracy: 1.0000 - val_loss: 0.1492 - val_accuracy: 0.9625
Epoch 25/60
10/10 [==============================] - 2s 211ms/step - loss: 0.0068 - acc
uracy: 1.0000 - val_loss: 0.1343 - val_accuracy: 0.9625
Epoch 26/60
10/10 [==============================] - 2s 214ms/step - loss: 0.0051 - acc
uracy: 1.0000 - val_loss: 0.1554 - val_accuracy: 0.9625
Epoch 27/60
10/10 [==============================] - 2s 213ms/step - loss: 0.0046 - acc
uracy: 1.0000 - val_loss: 0.1454 - val_accuracy: 0.9625
Epoch 28/60
10/10 [==============================] - 2s 210ms/step - loss: 0.0041 - acc
uracy: 1.0000 - val_loss: 0.1333 - val_accuracy: 0.9625
Epoch 29/60
10/10 [==============================] - 3s 344ms/step - loss: 0.0036 - acc
uracy: 1.0000 - val_loss: 0.1405 - val_accuracy: 0.9625
Epoch 30/60
10/10 [==============================] - 2s 215ms/step - loss: 0.0033 - acc
uracy: 1.0000 - val_loss: 0.1398 - val_accuracy: 0.9625
Epoch 31/60
10/10 [==============================] - 2s 214ms/step - loss: 0.0030 - acc
uracy: 1.0000 - val_loss: 0.1377 - val_accuracy: 0.9625
Epoch 32/60
10/10 [==============================] - 2s 211ms/step - loss: 0.0028 - acc
uracy: 1.0000 - val_loss: 0.1403 - val_accuracy: 0.9625
Epoch 33/60
10/10 [==============================] - 2s 216ms/step - loss: 0.0027 - acc
uracy: 1.0000 - val_loss: 0.1415 - val_accuracy: 0.9625
Epoch 34/60
10/10 [==============================] - 3s 300ms/step - loss: 0.0025 - acc
uracy: 1.0000 - val_loss: 0.1385 - val_accuracy: 0.9625
Epoch 35/60
10/10 [==============================] - 3s 252ms/step - loss: 0.0023 - acc
uracy: 1.0000 - val_loss: 0.1375 - val_accuracy: 0.9625
Epoch 36/60
10/10 [==============================] - 3s 296ms/step - loss: 0.0022 - acc
uracy: 1.0000 - val_loss: 0.1405 - val_accuracy: 0.9625
Epoch 37/60
10/10 [==============================] - 4s 359ms/step - loss: 0.0021 - acc
uracy: 1.0000 - val_loss: 0.1403 - val_accuracy: 0.9625
Epoch 38/60
```

```
10/10 [==============================] - 2s 211ms/step - loss: 0.0020 - acc
uracy: 1.0000 - val_loss: 0.1419 - val_accuracy: 0.9625
Epoch 39/60
10/10 [==============================] - 3s 337ms/step - loss: 0.0019 - acc
uracy: 1.0000 - val_loss: 0.1406 - val_accuracy: 0.9625
Epoch 40/60
10/10 [==============================] - 2s 211ms/step - loss: 0.0018 - acc
uracy: 1.0000 - val_loss: 0.1391 - val_accuracy: 0.9625
Epoch 41/60
10/10 [==============================] - 2s 210ms/step - loss: 0.0017 - acc
uracy: 1.0000 - val_loss: 0.1428 - val_accuracy: 0.9625
Epoch 42/60
10/10 [==============================] - 2s 213ms/step - loss: 0.0016 - acc
uracy: 1.0000 - val_loss: 0.1415 - val_accuracy: 0.9625
Epoch 43/60
10/10 [==============================] - 2s 212ms/step - loss: 0.0015 - acc
uracy: 1.0000 - val_loss: 0.1431 - val_accuracy: 0.9625
Epoch 44/60
10/10 [==============================] - 3s 312ms/step - loss: 0.0014 - acc
uracy: 1.0000 - val_loss: 0.1428 - val_accuracy: 0.9625
Epoch 45/60
10/10 [==============================] - 2s 231ms/step - loss: 0.0014 - acc
uracy: 1.0000 - val_loss: 0.1404 - val_accuracy: 0.9625
Epoch 46/60
10/10 [==============================] - 2s 212ms/step - loss: 0.0013 - acc
uracy: 1.0000 - val_loss: 0.1435 - val_accuracy: 0.9625
Epoch 47/60
10/10 [==============================] - 2s 213ms/step - loss: 0.0013 - acc
uracy: 1.0000 - val_loss: 0.1444 - val_accuracy: 0.9625
Epoch 48/60
10/10 [==============================] - 2s 213ms/step - loss: 0.0012 - acc
uracy: 1.0000 - val_loss: 0.1439 - val_accuracy: 0.9625
Epoch 49/60
10/10 [==============================] - 2s 218ms/step - loss: 0.0011 - acc
uracy: 1.0000 - val_loss: 0.1442 - val_accuracy: 0.9625
Epoch 50/60
10/10 [==============================] - 3s 333ms/step - loss: 0.0011 - acc
uracy: 1.0000 - val_loss: 0.1469 - val_accuracy: 0.9625
Epoch 51/60
10/10 [==============================] - 2s 215ms/step - loss: 0.0011 - acc
uracy: 1.0000 - val_loss: 0.1420 - val_accuracy: 0.9625
Epoch 52/60
10/10 [==============================] - 2s 211ms/step - loss: 0.0010 - acc
uracy: 1.0000 - val_loss: 0.1464 - val_accuracy: 0.9625
Epoch 53/60
10/10 [==============================] - 2s 213ms/step - loss: 9.7355e-04 -
accuracy: 1.0000 - val_loss: 0.1446 - val_accuracy: 0.9625
Epoch 54/60
10/10 [==============================] - 2s 213ms/step - loss: 9.3644e-04 -
accuracy: 1.0000 - val_loss: 0.1475 - val_accuracy: 0.9625
Epoch 55/60
10/10 [==============================] - 3s 321ms/step - loss: 8.9502e-04 -
accuracy: 1.0000 - val_loss: 0.1475 - val_accuracy: 0.9625
Epoch 56/60
10/10 [==============================] - 2s 238ms/step - loss: 8.6761e-04 -
accuracy: 1.0000 - val_loss: 0.1458 - val_accuracy: 0.9625
```

```
Epoch 57/60
10/10 [==============================] - 2s 209ms/step - loss: 8.3691e-04 -
accuracy: 1.0000 - val_loss: 0.1482 - val_accuracy: 0.9625
Epoch 58/60
10/10 [==============================] - 2s 209ms/step - loss: 8.0988e-04 -
accuracy: 1.0000 - val_loss: 0.1491 - val_accuracy: 0.9625
Epoch 59/60
10/10 [==============================] - 2s 223ms/step - loss: 7.7744e-04 -
accuracy: 1.0000 - val_loss: 0.1458 - val_accuracy: 0.9625
Epoch 60/60
10/10 [==============================] - 2s 224ms/step - loss: 7.4348e-04 -
accuracy: 1.0000 - val_loss: 0.1464 - val_accuracy: 0.9625
3/3 [==============================] - 0s 42ms/step - loss: 0.1464 - accura
cy: 0.9625
Out-sample Test accuracy for CNN: 0.9624999761581421
Time Comsumption: 203.26114439964294 sec.
```




The accuracies reached its limit after 20 of epoches, which is 96.25% of accuracy.