**Machine Learning**

**COCSC17**

**PRACTICAL FILE**

****

**Submitted by:-**  **Submitted to:-**

2020UCO1580 Prof. Shubhra Goyal

Anurag Verma

**INDEX**

|  |  |  |
| --- | --- | --- |
| **Serial No.** | **Title** | **Teacher’s Signature** |
| 1 | Implement Linear Regression algorithm for Head Brain dataset. |  |
| 2 | Implement Logistic Regression Algorithm on Diabetes Dataset |  |
| 3 | Implement Naïve Bayes Algorithm for Diabetes Dataset |  |
| 4 | Implement K-Nearest Neighbor Algorithm on Diabetes Dataset |  |
| 5 | Implement K-Means Clustering Algorithm on Mall Customers Dataset |  |
| 6 | Implement Decision Tree using Plant Iris Dataset |  |
| 7 | Implement CNN |  |

**PRACTICAL-1**

**AIM:**

Implement Linear Regression algorithm for Head Brain dataset.

**Code & Outputs:**

import pandas as pd

import numpy as np

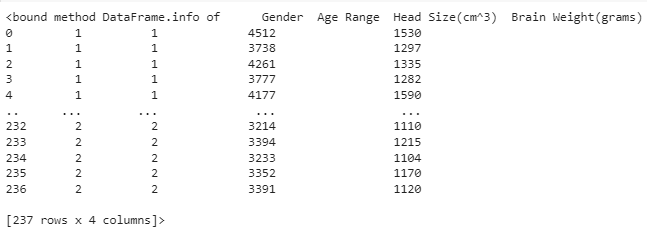
from matplotlib import pyplot as plt

df = pd.read\_csv('headbrain.csv')

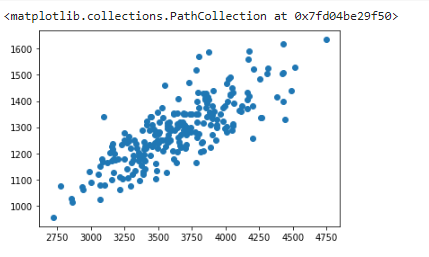
df.shape



df.info()



plt.scatter(df['Head Size(cm^3)'], df['Brain Weight(grams)'])



from sklearn import linear\_model

reg = linear\_model.LinearRegression()

reg.fit(df[['Head Size(cm^3)']], df['Brain Weight(grams)'])

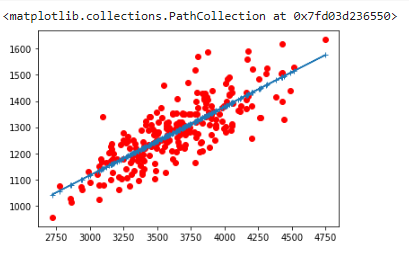


reg.predict([[3000]])



plt.plot(df['Head Size(cm^3)'], reg.predict(df[['Head Size(cm^3)']]), marker = '+')

plt.scatter(df['Head Size(cm^3)'],  df['Brain Weight(grams)'], color = 'red')



**PRACTICAL-2**

**AIM:**

Implement Logistic Regression Algorithm on Diabetes Dataset

**Code & Output:**

import numpy as np

import pandas as pd

import seaborn as sns

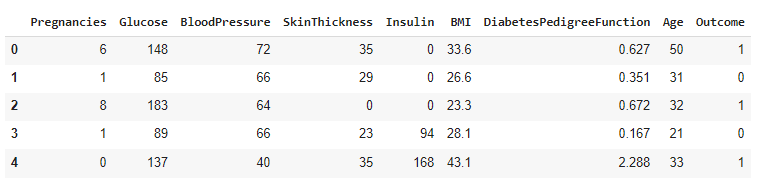
from matplotlib import pyplot as plt

%matplotlib inline

df = pd.read\_csv("diabetes.csv")

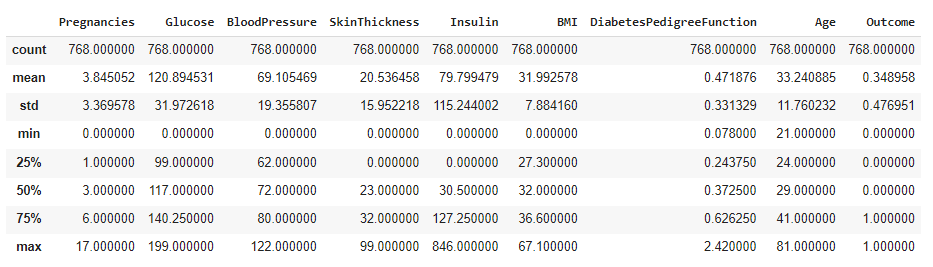
df.head()

# It will show top 5 rows



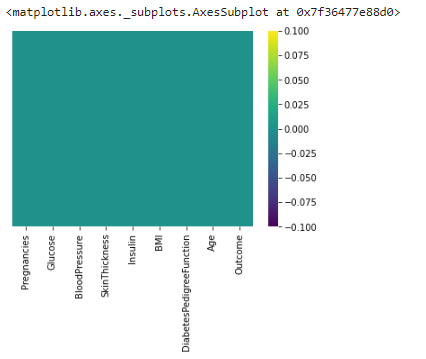
df.describe()

# It will show DataFrame



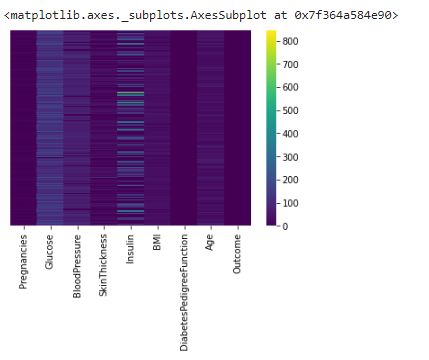
sns.heatmap(df.isnull(), yticklabels=False, cmap = 'viridis')

#It will plot heatmap graph with respect to no of null values



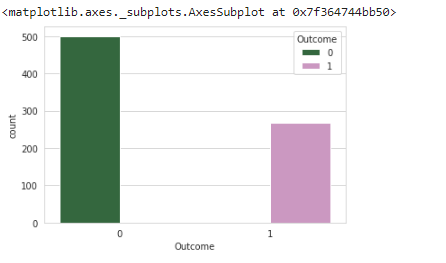
sns.heatmap(df, yticklabels = False, cmap = 'viridis')

#It will give heatmap graph with respect to values in the column



sns.set\_style('whitegrid')

sns.countplot(x='Outcome', hue = 'Outcome', data = df, palette = 'cubehelix')

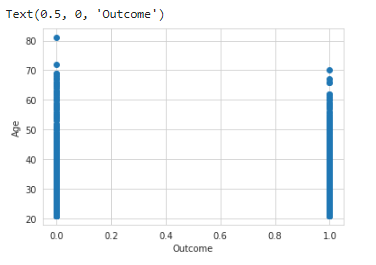


plt.scatter(x = 'Outcome', y = 'Age', data = df)

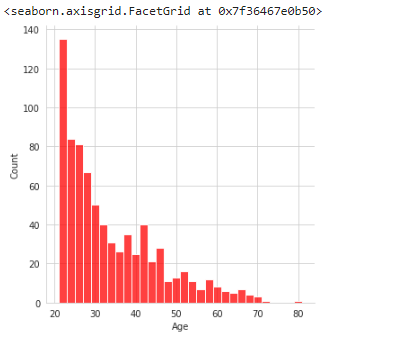
plt.ylabel('Age')

plt.xlabel('Outcome')

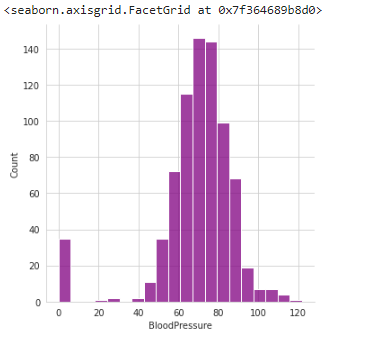
#It gives scattered(dotted) graph between Outcome and Age



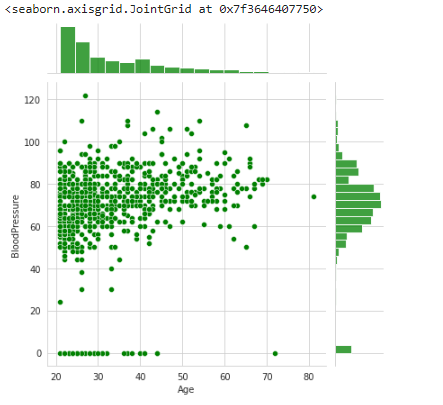
sns.displot(df['Age'], kde = False, color = 'red', bins = 30)



sns.displot(df['BloodPressure'], kde = False, color = 'purple', bins = 20)



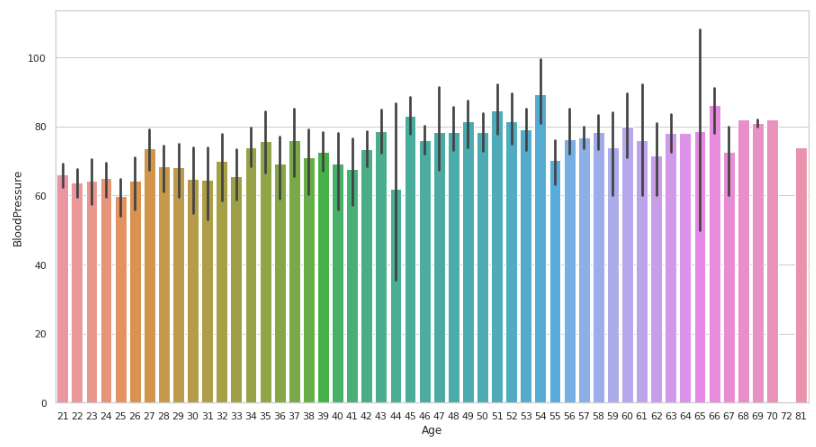
sns.jointplot(x = 'Age', y = 'BloodPressure', data = df, color = 'green')



sns.set(style = "whitegrid")

plt.figure(figsize = (15, 8))

ax = sns.barplot(x = 'Age', y = 'BloodPressure', data = df)



from sklearn.model\_selection import train\_test\_split

x = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']

y = ['Output']

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df.drop('Outcome', axis = 1), df['Outcome'], test\_size=0.20,random\_state=101)

from sklearn.linear\_model import LogisticRegression

LRModel = LogisticRegression(solver = 'lbfgs', max\_iter = 7500)

#max\_iter defines the maximum iterations while using LogisticRegression

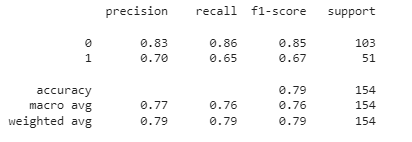
LRModel.fit(X\_train, y\_train)



predictions\_diabetes = LRModel.predict(X\_test)

from sklearn.metrics import classification\_report as cr, confusion\_matrix as cm

print(cr(y\_test, predictions\_diabetes))



x=['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','Age']

data = [0, 170, 126, 60, 35, 30.1, 0.649, 78]

pt = pd.DataFrame([data], columns = x)

pt.head()



predictions\_diabetes = LRModel.predict(pt)

print(predictions\_diabetes)



**PRACTICAL-3**

**AIM:**

Implement Naïve Bayes Algorithm for Diabetes Dataset

**Code & Output:**

import pandas as pd

import numpy as np

import seaborn as sns

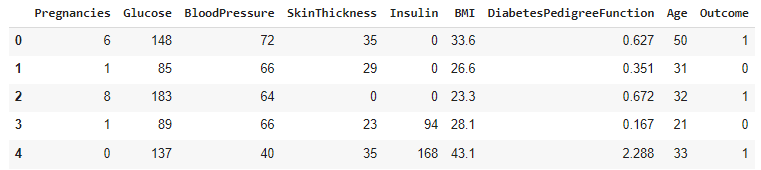
from sklearn.metrics import confusion\_matrix, f1\_score, accuracy\_score, classification\_report

from sklearn.naive\_bayes import GaussianNB

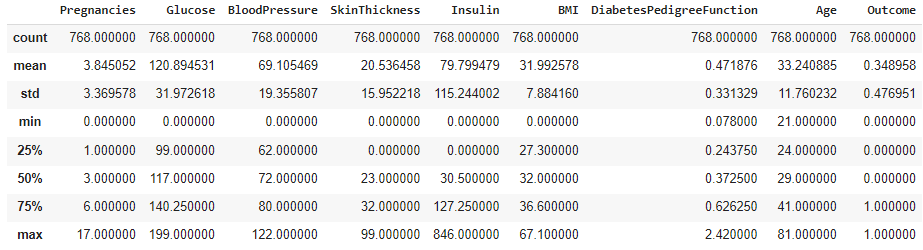
from sklearn.model\_selection import train\_test\_split

df = pd.read\_csv("diabetes.csv")

df.head()



df.describe()



x = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']

y = ['Output']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df.drop('Outcome', axis = 1), df['Outcome'], test\_size=0.20,random\_state=101)

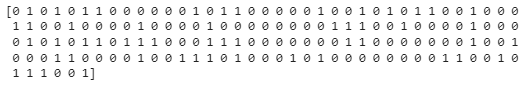
classifier\_Naive = GaussianNB()

classifier\_Naive.fit(X\_train, y\_train)



y\_pred = classifier\_Naive.predict(X\_test)

print(y\_pred)



x=['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','Age']

data = [0, 170, 126, 60, 35, 30.1, 0.649, 78]

pt = pd.DataFrame([data], columns = x)

pt.head()



#Pridicting on testing data

print(classifier\_Naive.predict(pt))



ac = accuracy\_score(y\_test,y\_pred)

cm = confusion\_matrix(y\_test, y\_pred)

print(ac)

print(cm)



**PRACTICAL-4**

**AIM:**

Implement K-Nearest Neighbor Algorithm on Diabetes Dataset

**Code & Output:**

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split #Used to train and test the data

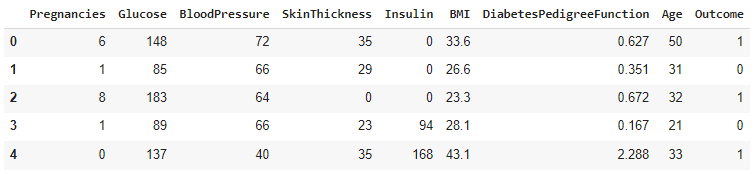
from sklearn.preprocessing import StandardScaler #From uniform data

from sklearn.neighbors import KNeighborsClassifier # for KNN

from sklearn.metrics import confusion\_matrix, f1\_score, accuracy\_score # For testing the data

df = pd.read\_csv('diabetes.csv')

df.head()



zero\_not\_accepted = ['Glucose', 'BloodPressure', 'SkinThickness', 'BMI', 'Insulin']

#We will replace zero values present in any of above column values

for column in zero\_not\_accepted:

  df[column] = df[column].replace(0, np.NaN)

  mean = int(df[column].mean(skipna = True))

  df[column] = df[column].replace(np.NaN, mean)

#Now we will split dataset into training data and testing data

X = df.iloc[:, 0:8]

#It means from all of the columns select 0->7 column as X-axis

y = df.iloc[:, 8]

#Select 8th column from all column as y-axis

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=0, test\_size = 0.2)

#It means 20% of the dataset will be used for testing purpose and rest for training purpose

#Scaling the data

sc\_X = StandardScaler()

X\_train = sc\_X.fit\_transform(X\_train)

X\_test = sc\_X.transform(X\_test)

classifier = KNeighborsClassifier(n\_neighbors = 11, p = 2, metric = 'euclidean')

#We have deckared KNN with k = 11 using euclidean distance

#p=2 is used because we're looking for the patient is diabetic or not

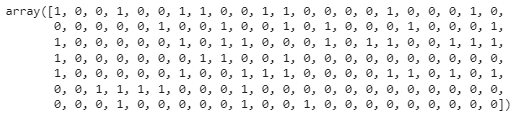
classifier.fit(X\_train, y\_train)



#Predicting the test results

y\_pred = classifier.predict(X\_test)

y\_pred



#Evaluating model

cm = confusion\_matrix(y\_test, y\_pred)

print(cm)

print(f1\_score(y\_test, y\_pred))



print(f1\_score(y\_test, y\_pred))



print(accuracy\_score(y\_test, y\_pred))



**PRACTICAL-5**

**AIM:**

Implement K-Means Clustering Algorithm on Mall Customers Dataset

**Code & Output:**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read\_csv('Mall\_Customers.csv')

X = dataset.iloc[:, [3, 4]].values

from sklearn.cluster import KMeans

wcss = []

for i in range(1, 11):

    kmeans = KMeans(n\_clusters = i, init = 'k-means++', random\_state = 42)

    kmeans.fit(X)

    wcss.append(kmeans.inertia\_)

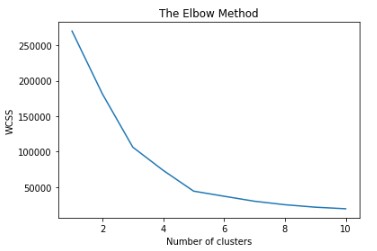
plt.plot(range(1, 11), wcss)

plt.title('The Elbow Method')

plt.xlabel('Number of clusters')

plt.ylabel('WCSS')

plt.show()



kmeans = KMeans(n\_clusters = 5, init = 'k-means++', random\_state = 42)

y\_kmeans = kmeans.fit\_predict(X)

plt.scatter(X[y\_kmeans == 0, 0], X[y\_kmeans == 0, 1], s = 100, c = 'red', label = 'Cluster 1')

plt.scatter(X[y\_kmeans == 1, 0], X[y\_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')

plt.scatter(X[y\_kmeans == 2, 0], X[y\_kmeans == 2, 1], s = 100, c = 'green', label = 'Cluster 3')

plt.scatter(X[y\_kmeans == 3, 0], X[y\_kmeans == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')

plt.scatter(X[y\_kmeans == 4, 0], X[y\_kmeans == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s = 300, c = 'yellow', label = 'Centroids')

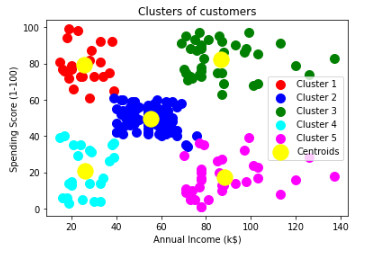
plt.title('Clusters of customers')

plt.xlabel('Annual Income (k$)')

plt.ylabel('Spending Score (1-100)')

plt.legend()

plt.show()



**PRACTICAL-6**

**AIM:**

Implement Decision Tree using Plant Iris Dataset

**Code & Ouput:**

import pandas as pd

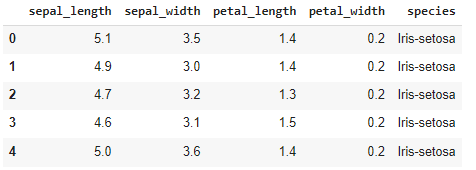
import numpy as np

from sklearn.tree import DecisionTreeClassifier

from sklearn import tree

df = pd.read\_csv('IRIS.csv')

df.head()



FEATURE\_NAMES = ['sepal\_length','sepal\_width', 'petal\_length', 'petal\_width']

X = df[FEATURE\_NAMES]

y = df['species']

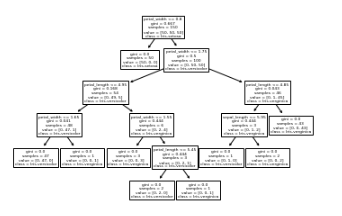
clf = DecisionTreeClassifier(max\_depth = 5)

clf.fit(X, y)



TV = ['Iris-setosa', 'Iris-versicolor', 'Iris-verginica']

tree.plot\_tree(clf, feature\_names = FEATURE\_NAMES, class\_names = TV);



**PRACTICAL-7**

**AIM:**

Implement CNN

**Code & Ouput:**

import tensorflow as tf

from tensorflow.keras import datasets, layers, models

import matplotlib.pyplot as plt

(train\_images, train\_labels), (test\_images, test\_labels) = datasets.cifar10.load\_data()

# Normalize pixel values to be between 0 and 1

train\_images, test\_images = train\_images / 255.0, test\_images / 255.0

class\_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',

'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(10,10))

for i in range(25):

plt.subplot(5,5,i+1)

plt.xticks([])

plt.yticks([])

plt.grid(False)

plt.imshow(train\_images[i])

# The CIFAR labels happen to be arrays,

# which is why you need the extra index

plt.xlabel(class\_names[train\_labels[i][0]])

plt.show()

model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)))

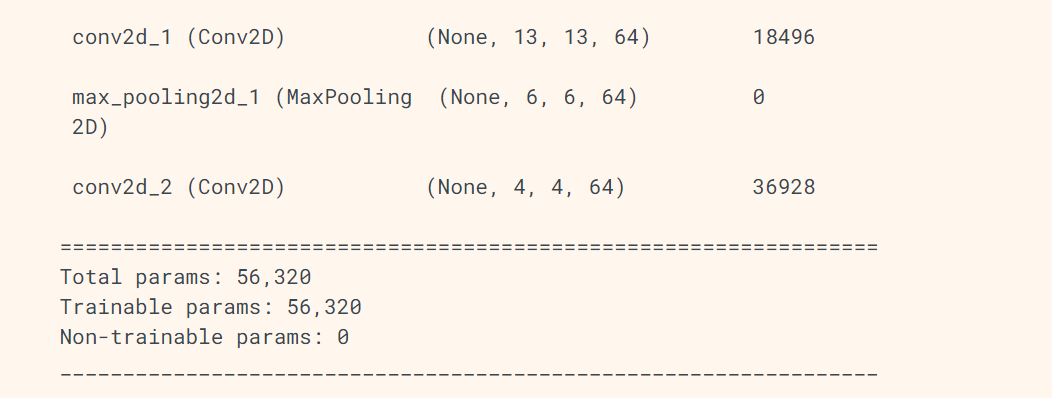
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.summary()

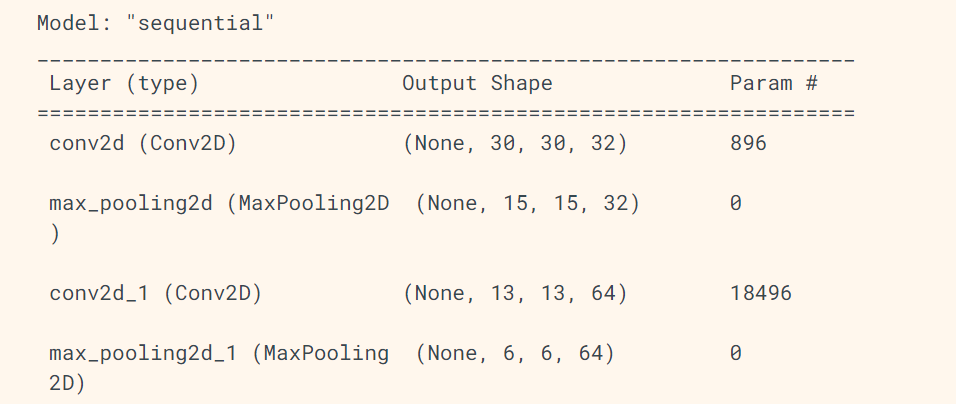


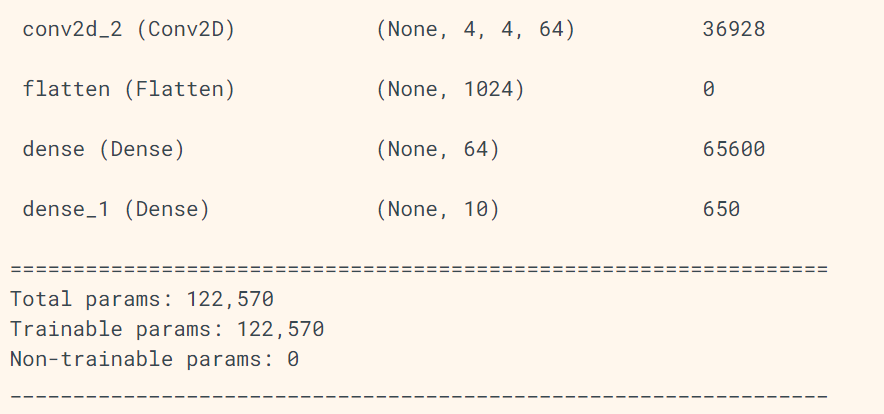
model.add(layers.Flatten())

model.add(layers.Dense(64, activation='relu'))

model.add(layers.Dense(10))

model.summary()



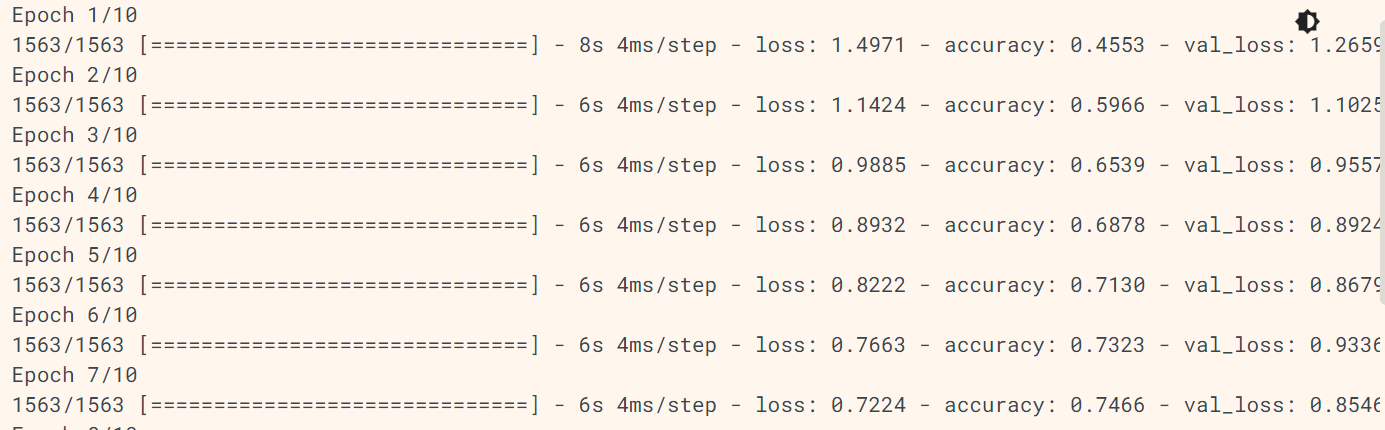


model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True),

metrics=['accuracy'])

history = model.fit(train\_images, train\_labels, epochs=10,



validation\_data=(test\_images, test\_labels))

plt.plot(history.history['accuracy'], label='accuracy')

plt.plot(history.history['val\_accuracy'], label = 'val\_accuracy')

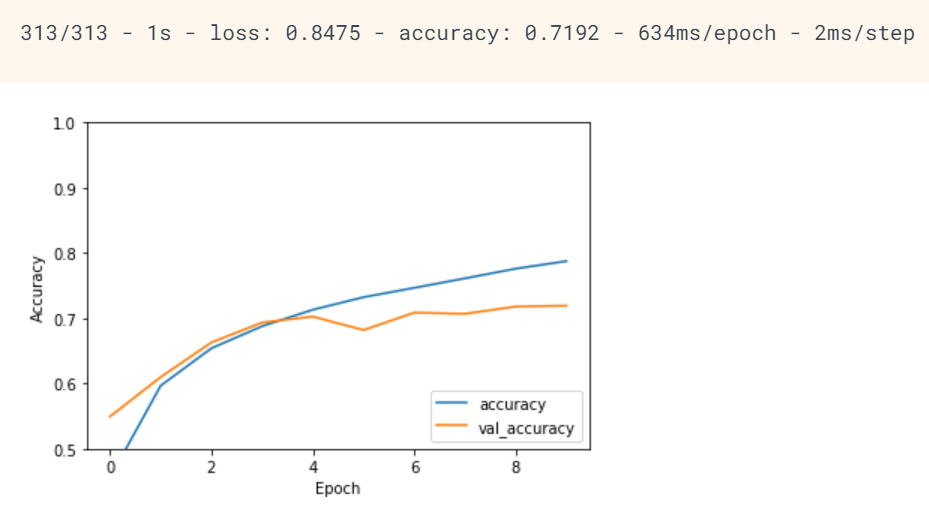
plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.ylim([0.5, 1])

plt.legend(loc='lower right')

test\_loss, test\_acc = model.evaluate(test\_images, test\_labels, verbose=2)



print(test\_acc)

