Assessment 1

Task 1: Regression

In [1]:
```python
# TensorFlow config to GPU
import tensorflow as tf
print(tf.__version__)

gpus = tf.config.list_physical_devices('GPU')
if gpus:
    tf.config.set_logical_device_configuration(
        gpus[0],
        [tf.config.LogicalDeviceConfiguration(memory_limit=15292)]
    )

logical_gpus = tf.config.list_logical_devices('GPU')
print(logical_gpus)
print(len(gpus), "Physical GPU,", len(logical_gpus), "Logical GPUs")



from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())


print()
print()

import tensorflow as tf
print("Num GPUs Available: ",
len(tf.config.list_physical_devices('GPU')))
```

```
2.6.0
[LogicalDevice(name='/device:GPU:0', device_type='GPU')]
1 Physical GPU, 1 Logical GPUs
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 2090261792891546744
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 16034824192
locality {
  bus_id: 1
  links {
  }
}
incarnation: 5913532685654759661
physical_device_desc: "device: 0, name: NVIDIA GeForce RTX 2060, pci bus id: 0000:0
1:00.0, compute capability: 7.5"
]
```

```
Num GPUs Available:  1
```

In [2]:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Data Preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import normalize

# Test Train Split
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold

# Build Model
from sklearn.linear_model import Lasso

# Model Evaluation
from sklearn.metrics import mean_squared_error, r2_score
```

Data

This data contains percentage of body fat, age, weight, height, and ten body circumferencemeasurements of 252 men. The purpose of this experiment is to explore if it would be possible to fitbody fat to other measurements using multiple regression, which could provide a convenient way ofestimating body fat for men using only a scale and a measuring tape.

In [3]:

```python
df = pd.read_csv('body_fat_data.csv')
df.shape
```

Out[3]:

```
(252, 18)
```

In [4]:

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 252 entries, 0 to 251
Data columns (total 18 columns):
 #   Column
Non-Null Count  Dtype
---  ------
--------------  -----
 0   Unnamed: 0
252 non-null    int64
 1   Percent body fat using Siri equation 495/Density
252 non-null    float64
 2   Density gm/cm^3
```

```
252 non-null    float64
 3   Age
252 non-null    int64
 4   Weight (lbs)
252 non-null    float64
 5   Height (inches)
252 non-null    float64
 6   Adiposity index = Weight/Height^2 (kg/m^2)
252 non-null    float64
 7   Fat Free Weight (1 - fraction of body fat) * Weight, using Brozek formula (lbs)
252 non-null    float64
 8   Neck circumference (cm)
252 non-null    float64
 9   Chest circumference (cm)
252 non-null    float64
 10  Abdomen circumference (cm)
252 non-null    float64
 11  Hip circumference (cm)
252 non-null    float64
 12  Thigh circumference (cm)
252 non-null    float64
 13  Knee circumference (cm)
252 non-null    float64
 14  Ankle circumference (cm)
252 non-null    float64
 15  Extended biceps circumference (cm)
252 non-null    float64
 16  Forearm circumference (cm)
252 non-null    float64
 17  Wrist circumference (cm)
252 non-null    float64
dtypes: float64(16), int64(2)
memory usage: 35.6 KB
```

As you can see, there are 252 rows and 18 columns in the data. The first column is the index column, in our case is not necessary. The second column is the percentage of body fat, which is the target variable. The rest of the columns are the predictors. The predictors are age, weight, height, Adioposity, Fat Free Weight, Neck circumference, Chest circumference, Abdomen circumference, Hip circumference, Thigh circumference, Knee circumference, Ankle circumference, Biceps circumference, Forearm circumference and Wrist circumference.

In [5]:

```
# Removing the first column and renaming the columns to make it easier
to work with
df = df.iloc[: , 1:]

df.rename(columns={'Percent body fat using Siri equation 495/Density':
'BodyFat',
                     'Density gm/cm^3': 'Density',
                    'Weight (lbs)': 'Weight',
                    'Height (inches)': 'Height',
                    'Adiposity index = Weight/Height^2 (kg/m^2)' :
'AdiposityIndex',
                   'Fat Free Weight (1 - fraction of body fat) * Weight,
using Brozek formula (lbs)': 'FatFreeWeight',
                    'Neck circumference (cm)': 'Neck',
                     'Chest circumference (cm)': 'Chest',
                     'Abdomen circumference (cm)': 'Abdomen',
                     'Hip circumference (cm)': 'Hip',
                     'Thigh circumference (cm)': 'Thigh',
                     'Knee circumference (cm)': 'Knee',
                     'Ankle circumference (cm)': 'Ankle',
                     'Extended biceps circumference (cm)':
'ExtendedBiceps',
                     'Forearm circumference (cm)': 'Forearm',
                     'Wrist circumference (cm)': 'Wrist'
                    }, inplace=True)
```

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 252 entries, 0 to 251
Data columns (total 17 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   BodyFat         252 non-null    float64
 1   Density         252 non-null    float64
 2   Age             252 non-null    int64  
 3   Weight          252 non-null    float64
 4   Height          252 non-null    float64
 5   AdiposityIndex  252 non-null    float64
 6   FatFreeWeight   252 non-null    float64
 7   Neck            252 non-null    float64
 8   Chest           252 non-null    float64
 9   Abdomen         252 non-null    float64
 10  Hip             252 non-null    float64
 11  Thigh           252 non-null    float64
 12  Knee            252 non-null    float64
 13  Ankle           252 non-null    float64
 14  ExtendedBiceps  252 non-null    float64
 15  Forearm         252 non-null    float64
 16  Wrist           252 non-null    float64
dtypes: float64(16), int64(1)
memory usage: 33.6 KB
```

In [7]:
```
df.shape
```

Out[7]:
```
(252, 17)
```

In [8]:
```
df.head()
```

Out[8]:

	BodyFat	Density	Age	Weight	Height	AdiposityIndex	FatFreeWeight	Neck	Chest	Abdomen
0	12.3	1.0708	23	154.25	67.75	23.7	134.9	36.2	93.1	85.2
1	6.1	1.0853	22	173.25	72.25	23.4	161.3	38.5	93.6	83.0
2	25.3	1.0414	22	154.00	66.25	24.7	116.0	34.0	95.8	87.9
3	10.4	1.0751	26	184.75	72.25	24.9	164.7	37.4	101.8	86.4
4	28.7	1.0340	24	184.25	71.25	25.6	133.1	34.4	97.3	100.0

In [9]:
```python
df.describe()
```

Out[9]:

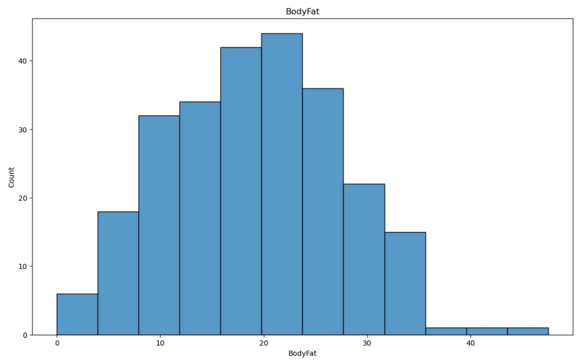
	BodyFat	Density	Age	Weight	Height	AdiposityIndex	FatFreeWeight
count	252.000000	252.000000	252.000000	252.000000	252.000000	252.000000	252.000000
mean	19.150794	1.055574	44.884921	178.924405	70.148810	25.436905	143.713889
std	8.368740	0.019031	12.602040	29.389160	3.662856	3.648111	18.231642
min	0.000000	0.995000	22.000000	118.500000	29.500000	18.100000	105.900000
25%	12.475000	1.041400	35.750000	159.000000	68.250000	23.100000	131.350000
50%	19.200000	1.054900	43.000000	176.500000	70.000000	25.050000	141.550000
75%	25.300000	1.070400	54.000000	197.000000	72.250000	27.325000	153.875000
max	47.500000	1.108900	81.000000	363.150000	77.750000	48.900000	240.500000

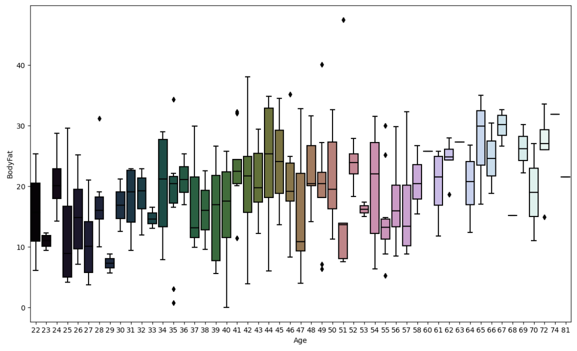
In [10]:
```python
df.isnull().sum()
```

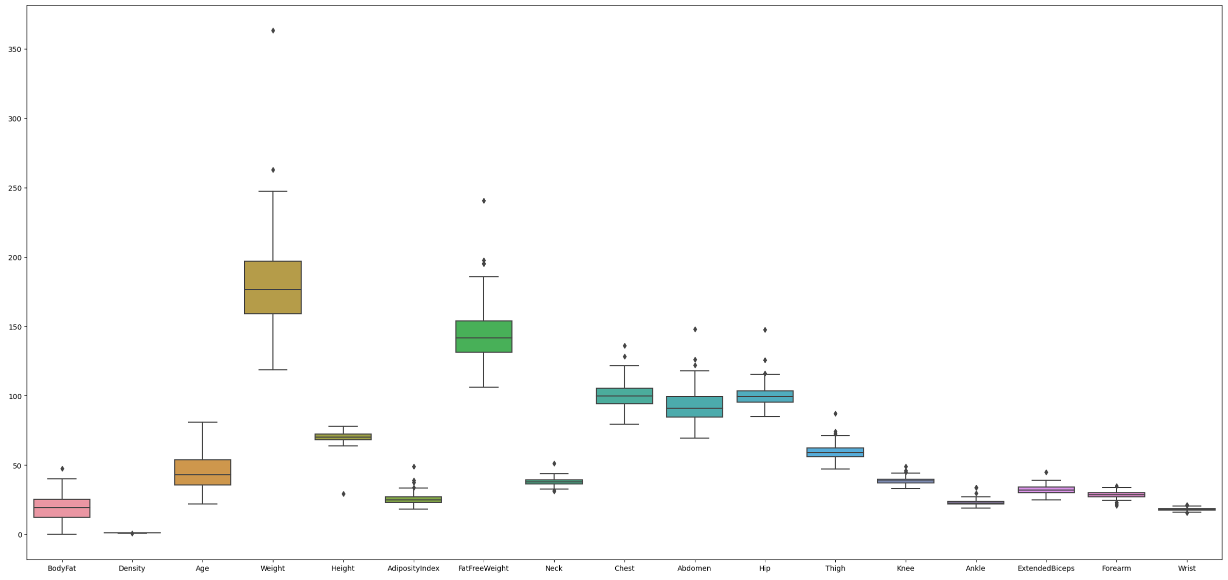
Out[10]:
```
BodyFat           0
Density           0
Age               0
Weight            0
Height            0
AdiposityIndex    0
FatFreeWeight     0
Neck              0
Chest             0
Abdomen           0
Hip               0
Thigh             0
Knee              0
Ankle             0
ExtendedBiceps    0
Forearm           0
Wrist             0
dtype: int64
```

There are no missing values in the data. The data is already cleaned but the predictors are not scaled. You need to scale the predictors before you fit the model.

In [11]:
```python
# Data Visualization
plt.figure(figsize=(30,8))


plt.subplot(1,2,1)
plt.title('BodyFat')
sns.histplot(df.BodyFat)


plt.subplot(1,2,2)
sns.boxplot(x=df.Age, y=df.BodyFat, palette=("cubehelix"))


plt.figure(figsize=(30,14))
sns.boxplot(data=df)
plt.show()
```

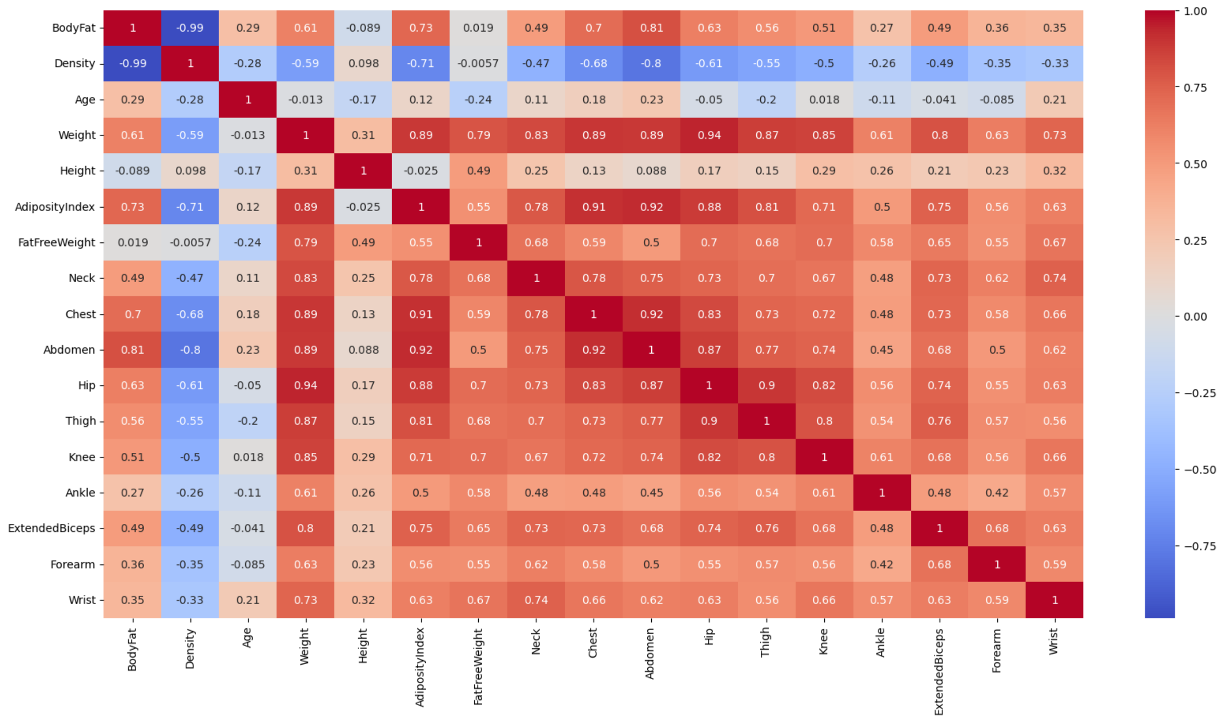






In [12]:

```
# Getting the correlation matrix
corr = df.corr()


# Plotting the correlation matrix
plt.figure(figsize=(20,10))
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.show()
```



In [13]:
```
# Creating X and y
X = df.drop('BodyFat',axis=1)
y = df['BodyFat']
```

In [14]:
```
X
```

Out[14]:

	Density	Age	Weight	Height	AdiposityIndex	FatFreeWeight	Neck	Chest	Abdomen	Hip
0	1.0708	23	154.25	67.75	23.7	134.9	36.2	93.1	85.2	94.5
1	1.0853	22	173.25	72.25	23.4	161.3	38.5	93.6	83.0	98.7
2	1.0414	22	154.00	66.25	24.7	116.0	34.0	95.8	87.9	99.2
3	1.0751	26	184.75	72.25	24.9	164.7	37.4	101.8	86.4	101.2
4	1.0340	24	184.25	71.25	25.6	133.1	34.4	97.3	100.0	101.9
...	...	...	...	...	...	...	...	...	...	...
247	1.0736	70	134.25	67.00	21.1	118.9	34.9	89.2	83.6	88.8
248	1.0236	72	201.00	69.75	29.1	136.1	40.9	108.5	105.0	104.5
249	1.0328	72	186.75	66.00	30.2	133.9	38.9	111.1	111.5	101.7
250	1.0399	72	190.75	70.50	27.0	142.6	38.9	108.3	101.3	97.8
251	1.0271	74	207.50	70.00	29.8	143.7	40.8	112.4	108.5	107.1

252 rows × 16 columns

In [15]:
```
y
```

Out[15]:
```
0       12.3
1        6.1
2       25.3
3       10.4
```

```
4      28.7
       ...
247    11.0
248    33.6
249    29.3
250    26.0
251    31.9
Name: BodyFat, Length: 252, dtype: float64
```

In [16]:
```python
# Standardize the data
scaler = StandardScaler()


X = normalize(X)
X = scaler.fit_transform(X)
# convert y to numpy array
y = np.array(y)
```

In [17]:
```python
y
```

Out[17]:
```
array([12.3,  6.1, 25.3, 10.4, 28.7, 20.9, 19.2, 12.4,  4.1, 11.7,  7.1,
        7.8, 20.8, 21.2, 22.1, 20.9, 29. , 22.9, 16. , 16.5, 19.1, 15.2,
       15.6, 17.7, 14. ,  3.7,  7.9, 22.9,  3.7,  8.8, 11.9,  5.7, 11.8,
       21.3, 32.3, 40.1, 24.2, 28.4, 35.2, 32.6, 34.5, 32.9, 31.6, 32. ,
        7.7, 13.9, 10.8,  5.6, 13.6,  4. , 10.2,  6.6,  8. ,  6.3,  3.9,
       22.6, 20.4, 28. , 31.5, 24.6, 26.1, 29.8, 30.7, 25.8, 32.3, 30. ,
       21.5, 13.8,  6.3, 12.9, 24.3,  8.8,  8.5, 13.5, 11.8, 18.5,  8.8,
       22.2, 21.5, 18.8, 31.4, 26.8, 18.4, 27. , 27. , 26.6, 14.9, 23.1,
        8.3, 14.1, 20.5, 18.2,  8.5, 24.9,  9. , 17.4,  9.6, 11.3, 17.8,
       22.2, 21.2, 20.4, 20.1, 22.3, 25.4, 18. , 19.3, 18.3, 17.3, 21.4,
       19.7, 28. , 22.1, 21.3, 26.7, 16.7, 20.1, 13.9, 25.8, 18.1, 27.9,
       25.3, 14.7, 16. , 13.8, 17.5, 27.2, 17.4, 20.8, 14.9, 18.1, 22.7,
       23.6, 26.1, 24.4, 27.1, 21.8, 29.4, 22.4, 20.4, 24.9, 18.3, 23.3,
        9.4, 10.3, 14.2, 19.2, 29.6,  5.3, 25.2,  9.4, 19.6, 10.1, 16.5,
       21. , 17.3, 31.2, 10. , 12.5, 22.5,  9.4, 14.6, 13. , 15.1, 27.3,
       19.2, 21.8, 20.3, 34.3, 16.5,  3. ,  0.7, 20.5, 16.9, 25.3,  9.9,
       13.1, 29.9, 22.5, 16.9, 26.6,  0. , 11.5, 12.1, 17.5,  8.6, 23.6,
       20.4, 20.5, 24.4, 11.4, 38.1, 15.9, 24.7, 22.8, 25.5, 22. , 17.7,
        6.6, 23.6, 12.2, 22.1, 28.7,  6. , 34.8, 16.6, 32.9, 32.8,  9.6,
       10.8,  7.1, 27.2, 19.5, 18.7, 19.5, 47.5, 13.6,  7.5, 24.5, 15. ,
       12.4, 26. , 11.5,  5.2, 10.9, 12.5, 14.8, 25.2, 14.9, 17. , 10.6,
       16.1, 15.4, 26.7, 25.8, 18.6, 24.8, 27.3, 12.4, 29.9, 17. , 35. ,
       30.4, 32.6, 29. , 15.2, 30.2, 11. , 33.6, 29.3, 26. , 31.9])
```

When we have multiple feaures having various degrees of magnitude, range, units and other statistical properties, it is a good practice to scale the features so that they can be uniformly evaluated. This is called feature scaling. There are many ways to scale the features. In this task, you will use the standardization method to scale the features. The standardization method subtracts the mean from each value and then divides the difference by the standard deviation. The result is a feature that has a mean of 0 and a standard deviation of 1.

In [18]:
```python
# Splitting the data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
y_train.shape
```

Out[18]:
```
(201,)
```

In [19]:
```python
# Using Lasso Regression
lasso = Lasso(alpha=10)
lasso.fit(X_train, y_train)

# Predicting the test set
y_pred = lasso.predict(X_test)

# Evaluating the model
print('Mean Squared Error:', mean_squared_error(y_test, y_pred))
print('R2 Score:', r2_score(y_test, y_pred))
```

```
Mean Squared Error: 48.50157579093122
R2 Score: -0.04263946932861762
```

For this task, we will use multiple linear regression to predict the percentage of body fat. More specifically, we will use Laso regression to fit the model. The reason we use Laso regression is that it can automatically select the most important features and remove the less important features. This is very useful when we have a lot of features and we want to reduce the dimensionality of the data.

From the above cell, we can see that for different alpha values, we get different MSE and R2 values. The MSE and R2 values are the performance metrics of the model. The MSE is the mean squared error, which is the average of the squared differences between the target and the prediction. The R2 is the coefficient of determination, which is a statistical measure of how well the regression predictions approximate the real data points. The R2 value is between 0 and 1. The higher the R2 value, the better the model fits the data.

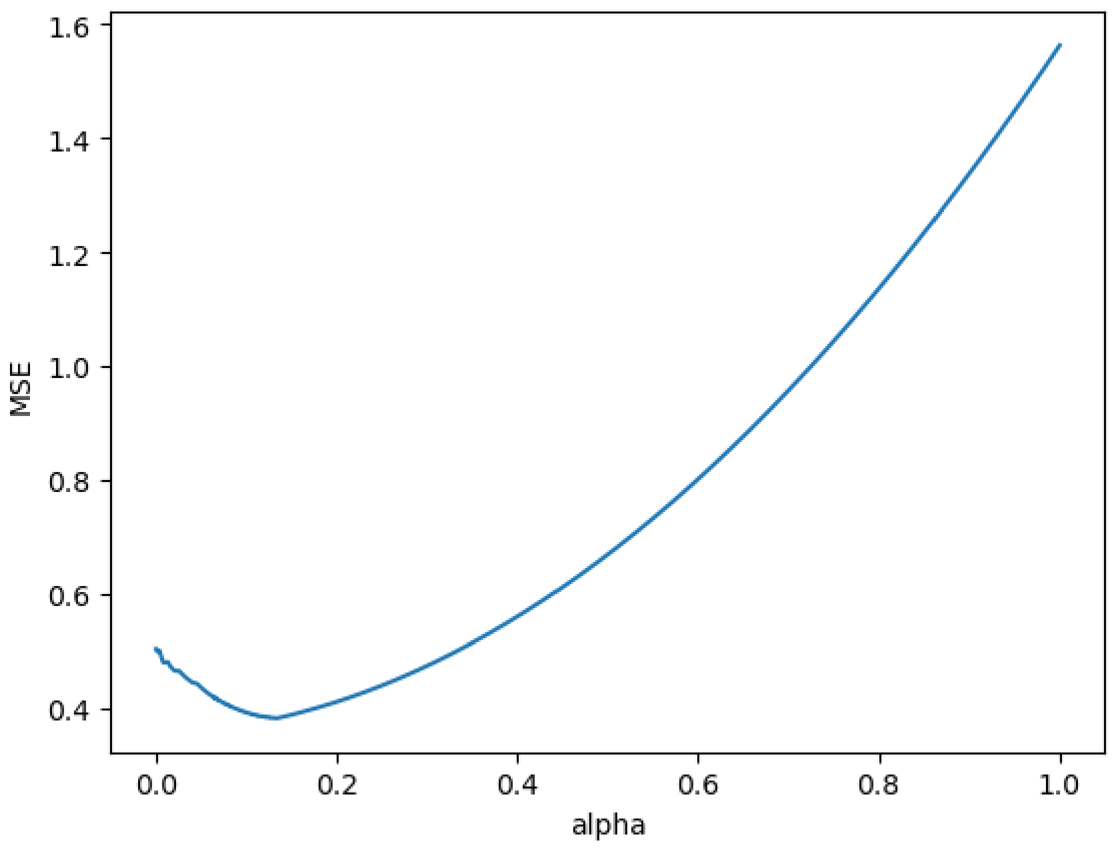
To find the best alpha (the hyper parameter of the model), we can use the cross validation method. The cross validation method splits the data into k folds. For each fold, we train the model on the rest of the data and evaluate the model on the fold. We repeat this process k times and get k MSE and R2 values. The average of the k MSE and R2 values is the average MSE and R2 values. The hyper parameter that gives the best average MSE and R2 values is the best hyper parameter.

In [20]:
```python
# Using K-Fold Cross Validation
def train_cv(X_sub, y_sub, n_splits=5, shuffle=True):
    k_fold = KFold(n_splits = n_splits, shuffle=shuffle)
    k_fold.get_n_splits(X_sub)
    curr_alpha = 0.00001 #### 0.00001 -> 1
    alpha_list = []
    MSE = []
    for train_index, test_index in k_fold.split(X_sub):
        X_train_sub, X_test_sub = X_sub[train_index], X_sub[test_index]
        y_train_sub, y_test_sub = y_sub[train_index], y_sub[test_index]

        # create a Lasso object
        reg_model = Lasso()

        # get each alpha value and train the model
        while curr_alpha <= 1:
            reg_model.set_params(alpha=curr_alpha)
            alpha_list.append(curr_alpha) # [0.00001, 0.00002, 0.00003
..., 1]
            reg_model.fit(X_train_sub, y_train_sub)
            y_pred_sub = reg_model.predict(X_test_sub)
            MSE.append(mean_squared_error(y_test_sub, y_pred_sub))
            # print("Mean squared error: %.2f" %
mean_squared_error(y_test, y_pred))
            curr_alpha += 0.00001
    return MSE, alpha_list
```

In [21]:
```
mse, alpha = train_cv(X_train,y_train)
```

Now we have our mean squared error and R2 values for different alpha values. We can plot the mean squared error and R2 values against the alpha values. The plot will show us the best alpha value.

In [22]:
```
# plot the MSE values for each alpha value
plt.plot(alpha, mse)
plt.xlabel('alpha')
plt.ylabel('MSE')
plt.show()
```



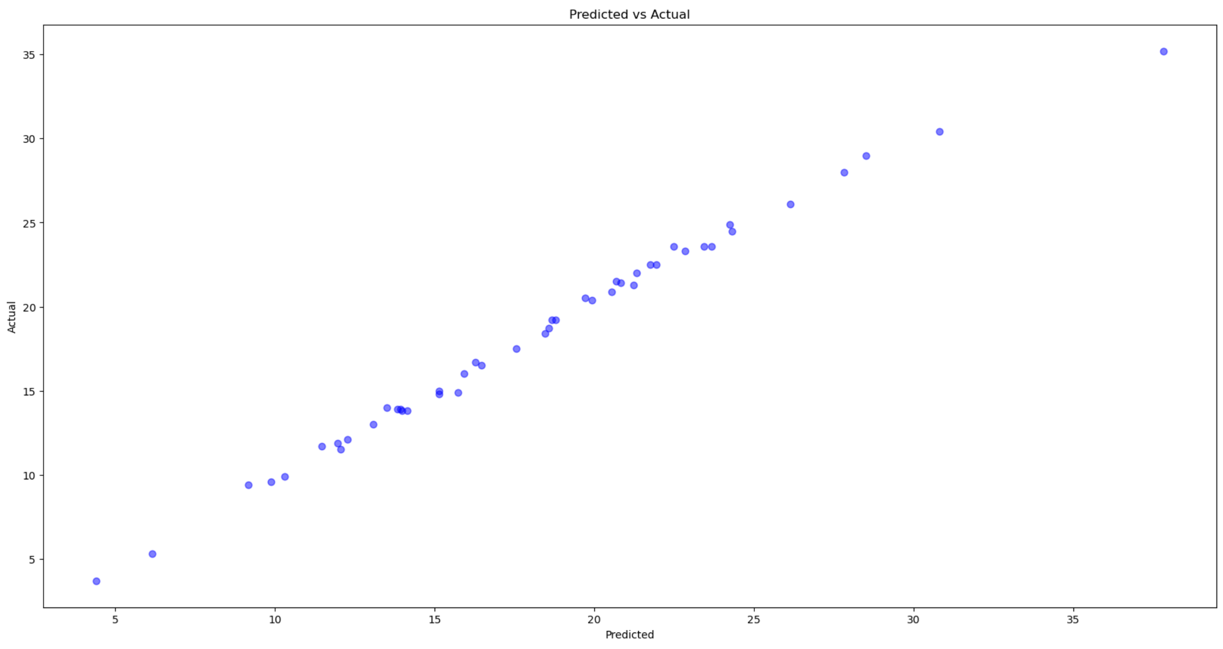
In [23]:
```
# Finding the best alpha value
min_mse = min(mse)
best_alpha = alpha[mse.index(min_mse)]
print('Best alpha value:', best_alpha)
```

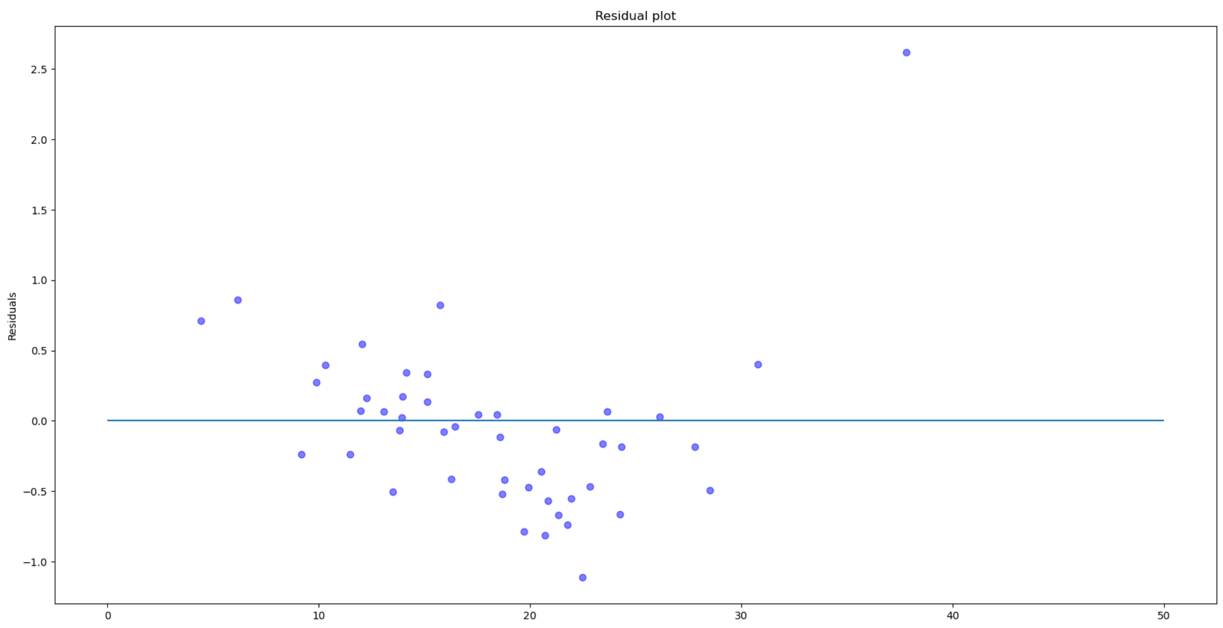
```
Best alpha value: 0.13322999999999247
```

In [24]:
```
# create a lasso regression with alpha value = best_alpha

X_train2, X_test2, y_train2, y_test2 = train_test_split(X_test, y_test,
test_size=0.9, random_state=42)

reg_model  = Lasso(alpha=best_alpha)
reg_model.fit(X_train, y_train)
y_pred_new = reg_model.predict(X_test2)

print(reg_model.alpha)

# print the mean squared error
print("Mean squared error: %.2f" % mean_squared_error(y_test2,
y_pred_new))

# print the R2 score
print('R2 score: %.2f' % r2_score(y_test2, y_pred_new))
```

```
0.13322999999999247
Mean squared error: 0.35
R2 score: 0.99
```

In [62]:
```python
# Plotting the predicted values vs the actual values
plt.figure(figsize=(20,10))
plt.scatter(y_pred_new, y_test2, c='b', s=40, alpha=0.5)
plt.title('Predicted vs Actual')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()


# Plotting the residuals
plt.figure(figsize=(20,10))
plt.scatter(y_pred_new, y_pred_new - y_test2, c='b', s=40, alpha=0.5)
plt.hlines(y=0, xmin=0, xmax=50)
plt.title('Residual plot')
plt.ylabel('Residuals')
plt.show()
```





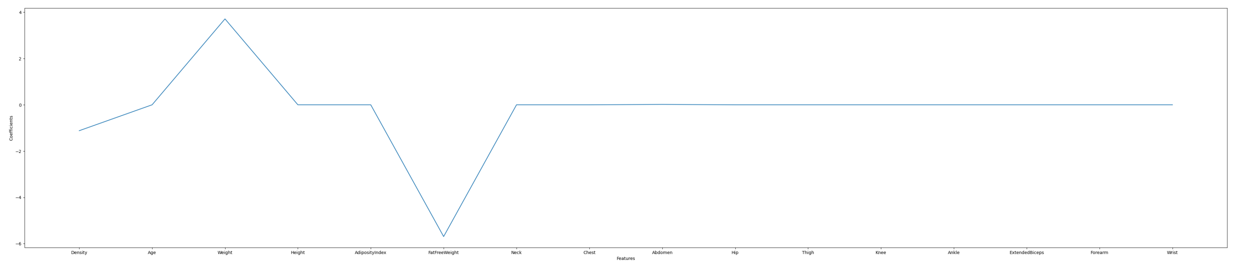
In [63]:
```
df_re = df.drop('BodyFat',axis=1)
list(df_re.columns)
```

Out[63]:
```
['Density',
 'Age',
 'Weight',
 'Height',
 'AdiposityIndex',
 'FatFreeWeight',
 'Neck',
 'Chest',
 'Abdomen',
 'Hip',
 'Thigh',
 'Knee',
 'Ankle',
 'ExtendedBiceps',
 'Forearm',
 'Wrist']
```

In [64]:
```
np.array(df_re.columns)[np.abs(reg_model.coef_) > 0]
```

Out[64]:
```
array(['Density', 'Weight', 'FatFreeWeight', 'Abdomen'], dtype=object)
```

In [65]:
```
# Plotting the coefficients
plt.figure(figsize=(50,10))
plt.plot(np.array(df_re.columns), reg_model.coef_)
plt.xlabel('Features')
plt.ylabel('Coefficients')
plt.show()
```



Features with the highest absolute coefficients are the most important features. The most important features are 'Density', 'Weight', 'FatFreeWeight', 'Abdomen'. The least important features are the ones with the lowest absolute coefficients.

- Density is the density of the body. The higher the density, the higher the percentage of body fat.
- Weight is the weight of the body. The higher the weight, the higher the percentage of body fat.
- FatFreeWeight is the weight of the body without fat. The higher the FatFreeWeight, the higher the percentage of body fat.
- Abdomen is the circumference of the abdomen. The higher the Abdomen, the higher the percentage of body fat.

After FatFreeWeight, the remaining features seems to have common trend. But when we look at the coefficients, we can see that the coefficient of Abdomen seems above zero. Lets check.

In [69]:
```
# coefficients of abdomen
reg_model.coef_[df_re.columns.get_loc('Abdomen')]
```

Out[69]:
```
0.016549619847761304
```

In [70]:
```
# coefficients of Thigh
reg_model.coef_[df_re.columns.get_loc('Thigh')]
```

Out[70]:
```
0.0
```

Thus we can say that Density, Weight, FatFreeWeight and Abdomen are the most important features. These features affect the percentage of body fat the most.

** --- **

Assessment 1

Task 2: Image classification

In [1]:

```python
# TensorFlow config to GPU
import tensorflow as tf
print(tf.__version__)

gpus = tf.config.list_physical_devices('GPU')
if gpus:
    tf.config.set_logical_device_configuration(
        gpus[0],
        [tf.config.LogicalDeviceConfiguration(memory_limit=15292)]
    )

logical_gpus = tf.config.list_logical_devices('GPU')
print(logical_gpus)
print(len(gpus), "Physical GPU,", len(logical_gpus), "Logical GPUs")



from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())

print()
print()

import tensorflow as tf
print("Num GPUs Available: ",
len(tf.config.list_physical_devices('GPU')))
```

```
2.6.0
[LogicalDevice(name='/device:GPU:0', device_type='GPU')]
1 Physical GPU, 1 Logical GPUs
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 11885025967862399815
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 16034824192
locality {
  bus_id: 1
  links {
  }
}
incarnation: 15088301213470768400
physical_device_desc: "device: 0, name: NVIDIA GeForce RTX 2060, pci bus id: 0000:0
1:00.0, compute capability: 7.5"
]


Num GPUs Available:  1
```

In [2]:

```python
import pathlib
import os
import matplotlib.pyplot as plt
import numpy as np

# import sequential model
from tensorflow.keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D,
Rescaling
from tensorflow.keras.regularizers import l2
```

In [3]:

```python
# function to plot the training/validation accuracies/losses.

def plot_learning(history):
    fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(15,4))
    axes[0].plot(history.history['loss'])
    axes[0].plot(history.history['val_loss'])
    #axes[0].grid()
    axes[0].legend(['loss','val_loss'])
    axes[1].plot(history.history['accuracy'])
    axes[1].plot(history.history['val_accuracy'])
    #axes[1].grid()
    axes[1].legend(['accuracy','val_accuracy'])
```

In [4]:

```python
train_directory = pathlib.Path("blood_cell_data/TRAIN")
test_directory = pathlib.Path("blood_cell_data/TEST")
```

In [5]:

```python
class_names = os.listdir(train_directory)
class_names
```

Out[5]:

```
['EOSINOPHIL', 'LYMPHOCYTE', 'MONOCYTE', 'NEUTROPHIL']
```

We have a dataset of blood cells. The dataset folder contains 2 subfolders, `TRAIN` and `TEST` . Each of those subfolder contains images of blood cells of 4 different classes: `EOSINOPHIL` , `LYMPHOCYTE` , `MONOCYTE` , `NEUTROPHIL` . The task is to build a classifier that can classify blood cells into one of the 4 classes using Convolutional Neural Networks.

In [6]:

```python
# Count the number of images in Train directory
```

```
total = 0
for item in os.listdir(train_directory):
    print(item, len(list(train_directory.glob(item + '/*'))))
    total += len(list(train_directory.glob(item + '/*')))

print("Total number of Training images: ", total)

print()

# Count the number of images in Test directory
total = 0
for item in os.listdir(test_directory):
    print(item, len(list(test_directory.glob(item + '/*'))))
    total += len(list(test_directory.glob(item + '/*')))

print("Total number of Test images: ", total)
```

```
EOSINOPHIL 2497
LYMPHOCYTE 2483
MONOCYTE 2478
NEUTROPHIL 2499
Total number of Training images:  9957

EOSINOPHIL 13
LYMPHOCYTE 6
MONOCYTE 4
NEUTROPHIL 48
Total number of Test images:  71
```

We have in total 9957 images in the training set and 71 images in the test set.

In [7]:
```
all_images = list(train_directory.glob("EOSINOPHIL/*"))
print(all_images[0])
img = plt.imread(all_images[0])
img.shape
```

```
blood_cell_data\TRAIN\EOSINOPHIL\_0_1169.jpeg
```

Out[7]:
```
(240, 320, 3)
```

Each image is 240x320 pixels in RGB.

CNN (Convolutional Neural Network) is a class of deep neural networks, most commonly applied to analyzing visual imagery. CNNs use a variation of multilayer perceptrons designed to require minimal preprocessing. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics. They have applications in image and video recognition, recommender systems, image classification, medical image analysis, natural language processing, and financial time series.

Since it is a deep neural network, it is very easy to overfit the data. To avoid overfitting, we will use data augmentation. Data augmentation is a technique to artificially increase the size of a training dataset by creating modified versions of images in the dataset. The modified images are created by a series of random transforms such as random horizontal flips, random vertical flips, random rotations, random translations, and more. This is a very useful technique to prevent overfitting. We will use the `ImageDataGenerator` class from Keras to perform data augmentation.

In [8]:
```python
# Create generators
train_generator = tf.keras.preprocessing.image.ImageDataGenerator(

preprocessing_function=tf.keras.applications.mobilenet_v2.preprocess_input

    validation_split=0.2
)

test_generator = tf.keras.preprocessing.image.ImageDataGenerator(

preprocessing_function=tf.keras.applications.mobilenet_v2.preprocess_input

)
```

In [9]:
```python
# Flow image data
train_images=train_generator.flow_from_directory(
    directory=train_directory,
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=32,
    shuffle=True,
    seed=42,
    subset='training'
)

val_images=train_generator.flow_from_directory(
    directory=train_directory,
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=32,
    shuffle=True,
    seed=42,
    subset='validation'
)
```

```
test_images=test_generator.flow_from_directory(
    directory=test_directory,
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=32,
    shuffle=False
)
```

```
Found 7968 images belonging to 4 classes.
Found 1989 images belonging to 4 classes.
Found 71 images belonging to 4 classes.
```

In [10]:

```
# get the number of classes
num_classes = len(class_names)

model = Sequential([Rescaling(1./1., input_shape=(224, 224,3))])

model.add(Conv2D(filters = 32, kernel_size = (3,3),padding = 'Same',
                 activation ='relu', input_shape = (120, 160, 32)))
model.add(MaxPool2D(pool_size=(2,2)))

model.add(Conv2D(filters = 32, kernel_size = (3,3),padding = 'Same',
activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2)))

model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2)))

model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2)))

model.add(Conv2D(filters = 128, kernel_size = (3,3),padding = 'Same',
activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2)))



model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.20))
model.add(Dense(128, activation='relu'))

model.add(Dense(num_classes, activation = "softmax"))

model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

In [11]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
rescaling (Rescaling)       (None, 224, 224, 3)       0
_________________________________________________________________
conv2d (Conv2D)             (None, 224, 224, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 112, 112, 32)     0
_________________________________________________________________
conv2d_1 (Conv2D)           (None, 112, 112, 32)      9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 56, 56, 32)       0
_________________________________________________________________
conv2d_2 (Conv2D)           (None, 56, 56, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 28, 28, 64)       0
_________________________________________________________________
conv2d_3 (Conv2D)           (None, 28, 28, 64)        36928
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 64)       0
_________________________________________________________________
conv2d_4 (Conv2D)           (None, 14, 14, 128)       73856
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 128)        0
_________________________________________________________________
flatten (Flatten)           (None, 6272)              0
_________________________________________________________________
dense (Dense)               (None, 256)               1605888
_________________________________________________________________
dropout (Dropout)           (None, 256)               0
_________________________________________________________________
dense_1 (Dense)             (None, 256)               65792
_________________________________________________________________
dropout_1 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_2 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_3 (Dense)             (None, 4)                 516
=================================================================
Total params: 1,844,516
Trainable params: 1,844,516
Non-trainable params: 0
_________________________________________________________________
```

A CNN is a type of neural network that is trained on image data. CNNs are used to classify images into predefined classes. CNNs are very effective in image classification tasks.

Creating a CNN involves the following steps:

- Convolution
- Pooling
- Flattening
- Full connection

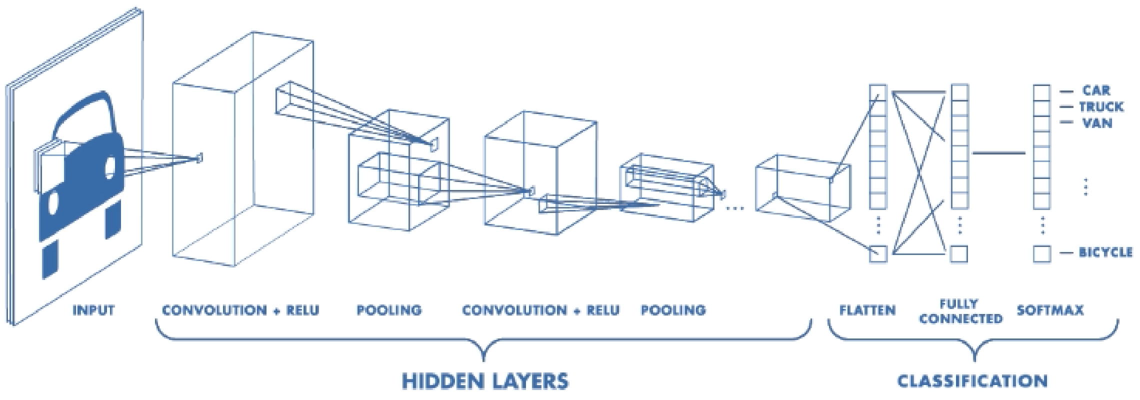
Convolution is the first step in a CNN. In this step, we create feature maps. A feature map is the result of a convolution operation between an input image and a filter. A filter is a small matrix that is used to extract features from an image. The filter is applied to the image by sliding it across the image. The filter is applied to every part of the image. The result of the convolution operation is called a feature map. The filter is applied to the image multiple times to create multiple feature maps. The number of feature maps created is equal to the number of filters used. The feature maps are stacked together to form a single tensor, which is called the output of the convolution operation. The output of the convolution operation is also called the activation map. The activation map is the result of applying the ReLU activation function to the convolution output.

Pooling is the next step in a CNN. In this step, we reduce the spatial size of the feature maps created in the convolution step. Pooling is performed on each feature map separately. The most common type of pooling operation is max pooling. In max pooling, we slide a pooling filter (also called pooling window) across the feature map and select the maximum value within the window. The result of the max pooling operation is a pooled feature map. The pooling filter is applied to every part of the feature map to create the pooled feature map. The size of the pooling filter is usually 2x2. The most common type of pooling filter is a 2x2 filter. The stride of the pooling filter is usually equal to the size of the filter. The most common stride is 2. The result of the max pooling operation is a feature map that is 1/4th the size of the original feature map.

Flattening is the third step in a CNN. In this step, we convert the pooled feature maps into a long vector that will be the input of the fully connected layers in the neural network. The flattened vector is a 1D vector that contains all the information about the image. The flattened vector is passed as input to the fully connected layers of the neural network.

Full connection is the last step in a CNN. In this step, we add one or more fully connected layers to the neural network. The fully connected layers are used to classify the images. The output of the fully connected layers is the result of the CNN.

The following diagram shows the architecture of a CNN:



Since we are using the Keras library, we don't have to implement the CNN from scratch. We can use the `Sequential` class to initialize the neural network. We can add the different layers of the CNN one by one using the `add` method. We can add the convolutional layer using the `Conv2D` class. We can add the pooling layer using the `MaxPooling2D` class. We can add the flattening layer using the `Flatten` class. We can add the fully connected layer using the `Dense` class. We can add the output layer using the `Dense` class. We can compile the CNN using the `compile` method. We can train the CNN on the training set and evaluate it on the test set using the `fit` and `evaluate` methods.

The `Conv2D` class creates a convolutional layer. The first parameter is the number of feature detectors (also called feature maps or feature channels). The second parameter is the shape of the feature detector. The third parameter is the input shape of the images. The fourth parameter is the activation function. We will use the `relu` activation function. The `relu` activation function is the most common activation function used in CNNs. The `relu` activation function is defined as:

$$f(x) = max(0, x)$$

where x is the input to the activation function.

The `MaxPooling2D` class creates a pooling layer. The first parameter is the size of the pooling filter. The second parameter is the stride of the pooling filter. The most common size of the pooling filter is 2x2. The most common stride is 2.

The `Flatten` class creates a flattening layer. This layer is used to convert the pooled feature maps into a long vector that will be the input of the fully connected layers.

The `Dense` class creates a fully connected layer in the neural network. The first parameter is the number of nodes in the fully connected layer. The second parameter is the activation function. We will use the `relu` activation function. The `relu` activation function is the most common activation function used in CNNs. The `relu` activation function is defined as:

$$f(x) = max(0, x)$$

where x is the input to the activation function.

The `Dense` class creates the output layer of the neural network. The first parameter is the number of nodes in the output layer. Since we have 4 classes, the number of nodes in the output layer is 4. The second parameter is the activation function. We will use the `softmax` activation function. The `softmax` activation function is used in the output layer of a neural network with more than 2 classes. The `softmax` activation function is defined as:

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

where x_i is the input to the activation function and n is the number of nodes in the output layer.

The `compile` method compiles the CNN. The first parameter is the optimizer. We will use the `adam` optimizer. The `adam` optimizer is an advanced version of the stochastic gradient descent algorithm. The `adam` optimizer is very efficient and effective. The `adam` optimizer is the most common optimizer used to train CNNs. The second parameter is the loss function. We will use the `categorical_crossentropy` loss function. The `categorical_crossentropy` loss function is the most common loss function used to train CNNs. The `categorical_crossentropy` loss function is defined as:

$$L = -\sum_{i=1}^{n} y_i \log(p_i)$$

where y_i is the target value and p_i is the predicted value.

The `fit` method trains the CNN on the training set. The first parameter is the training set. The second parameter is the target values of the training set. The third parameter is the number of epochs. We will train the CNN for 25 epochs. The fourth parameter is the batch size. We will use a batch size of 32. The `evaluate` method evaluates the performance of the CNN on the test set. The first parameter is the test set. The second parameter is the target values of the test set.

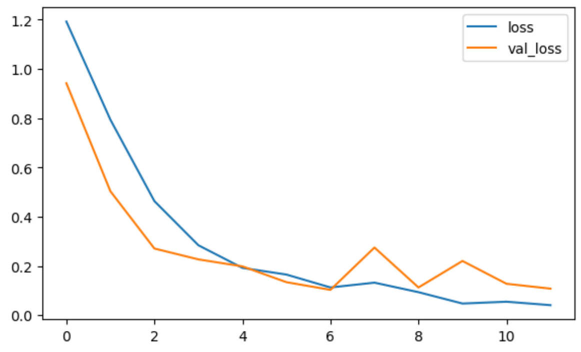
Now, since we are training the model with 100 epochs, it will take a lot of time to train the model. This can be solved using Early Stopping. Early Stopping is a technique used to stop the training of the model when the performance of the model on the validation set starts to decrease. We can use the `EarlyStopping` class to implement Early Stopping. The `EarlyStopping` class has the following parameters:

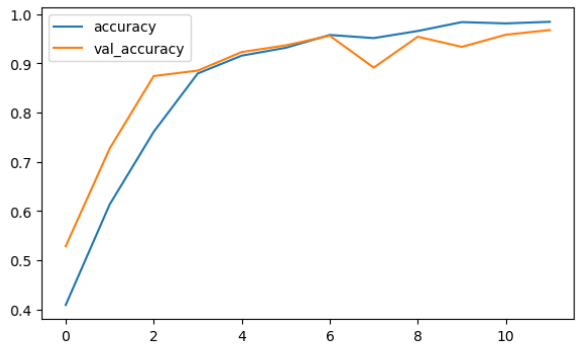
- `monitor` : The quantity to be monitored. We will monitor the `val_loss` quantity.
- `patience` : The number of epochs with no improvement after which training will be stopped. We will use a patience of 5.
- `restore_best_weights` : Whether to restore model weights from the epoch with the best value of the monitored quantity. We will use `True` .

In [12]:
```
history = model.fit(
    train_images,
    validation_data=val_images,
    epochs=100,
    callbacks=[
        tf.keras.callbacks.EarlyStopping(
            monitor='val_loss',
            patience=   5,
            restore_best_weights=True
        )
    ]
)
```

```
Epoch 1/100
249/249 [==============================] - 29s 99ms/step - loss: 1.1925 - accuracy:
0.4090 - val_loss: 0.9422 - val_accuracy: 0.5284
Epoch 2/100
249/249 [==============================] - 24s 96ms/step - loss: 0.7941 - accuracy:
0.6133 - val_loss: 0.5036 - val_accuracy: 0.7270
Epoch 3/100
249/249 [==============================] - 23s 92ms/step - loss: 0.4634 - accuracy:
0.7609 - val_loss: 0.2709 - val_accuracy: 0.8743
Epoch 4/100
249/249 [==============================] - 30s 121ms/step - loss: 0.2841 - accuracy:
0.8795 - val_loss: 0.2268 - val_accuracy: 0.8854
Epoch 5/100
249/249 [==============================] - 31s 124ms/step - loss: 0.1923 - accuracy:
0.9158 - val_loss: 0.1985 - val_accuracy: 0.9231
Epoch 6/100
249/249 [==============================] - 29s 116ms/step - loss: 0.1652 - accuracy:
0.9317 - val_loss: 0.1343 - val_accuracy: 0.9367
Epoch 7/100
249/249 [==============================] - 27s 108ms/step - loss: 0.1128 - accuracy:
0.9578 - val_loss: 0.1031 - val_accuracy: 0.9563
Epoch 8/100
249/249 [==============================] - 30s 119ms/step - loss: 0.1324 - accuracy:
0.9514 - val_loss: 0.2750 - val_accuracy: 0.8914
Epoch 9/100
249/249 [==============================] - 42s 160ms/step - loss: 0.0936 - accuracy:
0.9659 - val_loss: 0.1128 - val_accuracy: 0.9542
Epoch 10/100
249/249 [==============================] - 28s 114ms/step - loss: 0.0475 - accuracy:
0.9839 - val_loss: 0.2204 - val_accuracy: 0.9336
Epoch 11/100
249/249 [==============================] - 33s 131ms/step - loss: 0.0549 - accuracy:
0.9813 - val_loss: 0.1278 - val_accuracy: 0.9583
Epoch 12/100
249/249 [==============================] - 25s 101ms/step - loss: 0.0410 - accuracy:
0.9846 - val_loss: 0.1080 - val_accuracy: 0.9678
```

In [13]:
```
plot_learning(history)
```





In [14]:
```
# Evaluate the model on the test data using `evaluate`
print("Evaluate on test data")
```

```python
results = model.evaluate(test_images, batch_size=128)
print("test loss, test acc:", results)

# Generate predictions (probabilities -- the output of the last layer)
# on new data using `predict`

print()
print("Generate predictions")
predictions = model.predict(test_images[0])
print("predictions shape:", predictions.shape)
print("actual shape : ", test_images[0][1].shape)
```

```
Evaluate on test data
3/3 [==============================] - 0s 135ms/step - loss: 0.4362 - accuracy: 0.87
32
test loss, test acc: [0.4361788332462311, 0.8732394576072693]

Generate predictions
predictions shape: (32, 4)
actual shape :  (32, 4)
```