In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call dri
ve.mount("/content/drive", force_remount=True).
```

In [ ]:
```
from keras.datasets import mnist
from matplotlib import pyplot
```

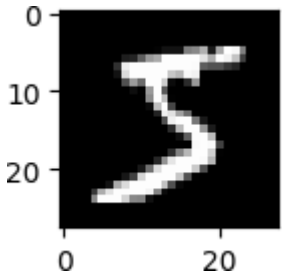
In [ ]:
```
# loading
(train_X, train_y), (test_X, test_y) = mnist.load_data()
```

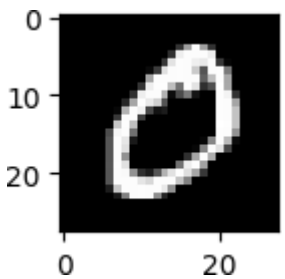
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-dataset
s/mnist.npz
11490434/11490434 [==============================] - 2s 0us/step
```

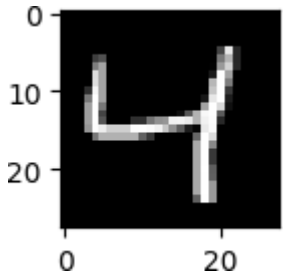
In [ ]:
```
# shape of dataset
print('X_train: ' + str(train_X.shape))
print('Y_train: ' + str(train_y.shape))
print('X_test: ' + str(test_X.shape))
print('Y_test: ' + str(test_y.shape))
```

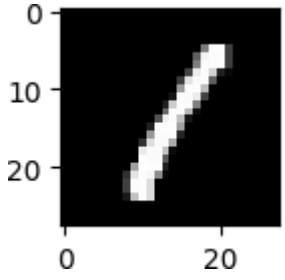
```
X_train: (60000, 28, 28)
Y_train: (60000,)
X_test: (10000, 28, 28)
Y_test: (10000,)
```

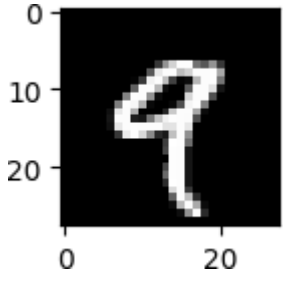
In [ ]:
```
# plotting
from matplotlib import pyplot
for i in range(9):
    pyplot.subplot(330 + 1 + i)
    pyplot.imshow(train_X[i], cmap=pyplot.get_cmap('gray'))
    pyplot.show()
```

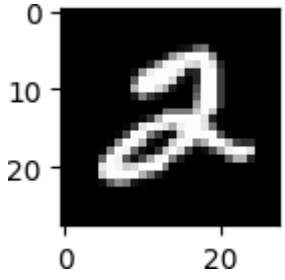


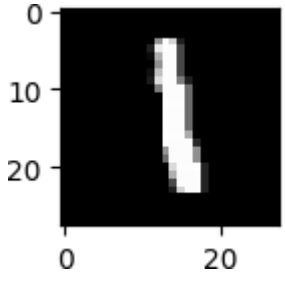


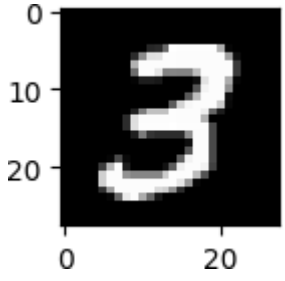

0
10
20
0
20

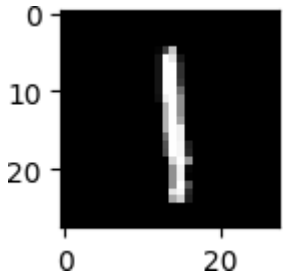

0
10
20
0
20


0
10
20
0
20


0
10
20
0
20


0
10
20
0
20


0
10
20
0
20


0
10
20
0
20

In [1]:
```
# Implementing feedforward neural networks with Keras and TensorFlow
# import the necessary packages
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import matplotlib.pyplot as plt
```

In [2]:
```
# grab the MNIST dataset (if this is your first time using this
# dataset then the 11MB download may take a minute)

print("[INFO] accessing MNIST...")
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape((x_train.shape[0], -1)).astype('float32') / 255
x_test = x_test.reshape((x_test.shape[0], -1)).astype('float32') / 255
```

```
[INFO] accessing MNIST...
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-dataset
s/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

In [4]:
```
# One-hot encode the labels
y_train = tf.keras.utils.to_categorical(y_train, num_classes=10)
y_test = tf.keras.utils.to_categorical(y_test, num_classes=10)


# Each data point in the MNIST dataset has an integer label in the range [0, 9],
# A label with a value of 0 indicates that the corresponding image contains a ze
# that the corresponding image contains the number eight.
```

In [ ]:
```
# However, we first need to transform these integer labels into vector labels,
#where the index in the vector for label is set to 1 and 0 otherwise (this proce
```

one-hot encoding representations for each digit, 0−9, in the listing below: 0: [1, 0, 0, 0, 0, 0, 0, 0, 0, 0] 1: [0, 1, 0, 0, 0, 0, 0, 0, 0, 0] 2: [0, 0, 1, 0, 0, 0, 0, 0, 0, 0] 3: [0, 0, 0, 1, 0, 0, 0, 0, 0, 0] 4: [0, 0, 0, 0, 1, 0, 0, 0, 0, 0] 5: [0, 0, 0, 0, 0, 1, 0, 0, 0, 0] 6: [0, 0, 0, 0, 0, 0, 1, 0, 0, 0] 7: [0, 0, 0, 0, 0, 0, 0, 1, 0, 0] 8: [0, 0, 0, 0, 0, 0, 0, 0, 1, 0] 9: [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]

In [7]:
```
# Step 3: Define the network architecture
model = Sequential([
    Dense(64, activation='relu', input_shape=(784,)),
    Dense(64, activation='relu'),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')
])
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87:  UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using S
equential models, prefer using an `Input(shape)` object as the first layer in the
model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [8]:
```
# Step 4: Compile the model
model.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accura
```

In [9]:
```
# Step 5: Train the model
H = model.fit(x_train, y_train, epochs=15, batch_size=32, validation_data=(x_tes
```

```
Epoch 1/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 5ms/step - accuracy: 0.5761 - loss: 1.3276 - v
al_accuracy: 0.9113 - val_loss: 0.3130
Epoch 2/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9084 - loss: 0.3089 - va
l_accuracy: 0.9315 - val_loss: 0.2393
Epoch 3/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9306 - loss: 0.2326 - va
l_accuracy: 0.9420 - val_loss: 0.2036
Epoch 4/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.9425 - loss: 0.1965 - v
al_accuracy: 0.9446 - val_loss: 0.1831
Epoch 5/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.9496 - loss: 0.1711 - v
al_accuracy: 0.9520 - val_loss: 0.1617
Epoch 6/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.9560 - loss: 0.1479 - va
l_accuracy: 0.9574 - val_loss: 0.1467
Epoch 7/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 4ms/step - accuracy: 0.9607 - loss: 0.1363 - va
l_accuracy: 0.9617 - val_loss: 0.1352
Epoch 8/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9638 - loss: 0.1231 - va
l_accuracy: 0.9612 - val_loss: 0.1261
Epoch 9/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9675 - loss: 0.1101 - va
l_accuracy: 0.9618 - val_loss: 0.1214
Epoch 10/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9681 - loss: 0.1066 - va
l_accuracy: 0.9651 - val_loss: 0.1174
Epoch 11/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9730 - loss: 0.0913 - v
al_accuracy: 0.9671 - val_loss: 0.1104
Epoch 12/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step - accuracy: 0.9741 - loss: 0.0870 - va
l_accuracy: 0.9658 - val_loss: 0.1125
Epoch 13/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9762 - loss: 0.0814 - v
al_accuracy: 0.9694 - val_loss: 0.1022
Epoch 14/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9780 - loss: 0.0756 - va
l_accuracy: 0.9674 - val_loss: 0.1073
Epoch 15/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 6ms/step - accuracy: 0.9786 - loss: 0.0726 - v
al_accuracy: 0.9656 - val_loss: 0.1155
```

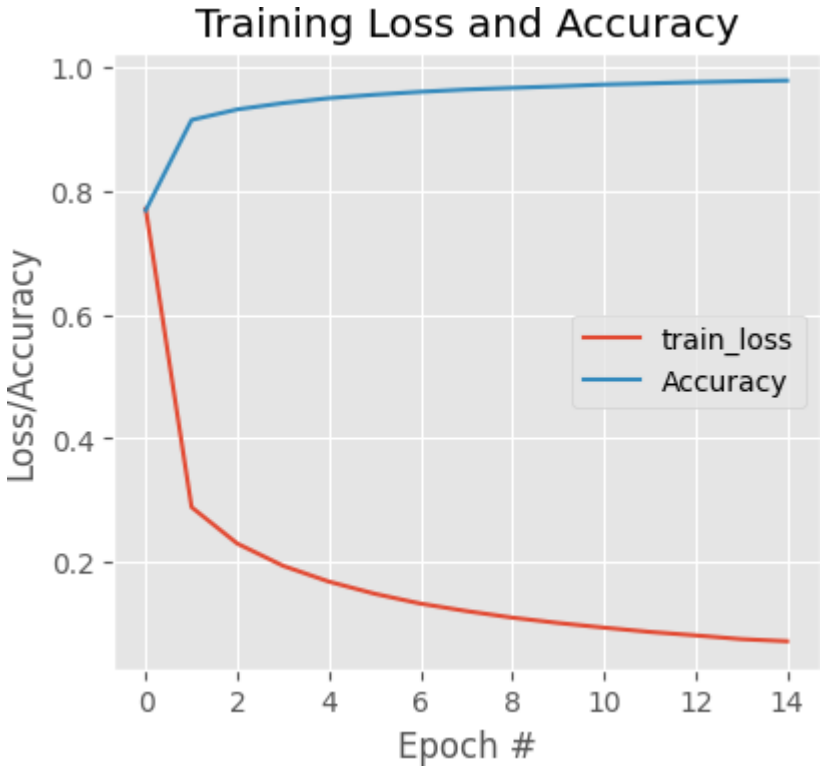
In [10]:
```
# Step 6: Evaluate the network
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_accuracy*100:.2f}%')
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9608 - loss: 0.1332
Test accuracy: 96.56%
```

In [11]:
```
# Step 7: Plot the training loss and accuracy
plt.style.use("ggplot")
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
```

```
plt.plot(H.history['loss'], label="train_loss")
plt.plot(H.history['accuracy'], label="Accuracy")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
```

Out[11]: <matplotlib.legend.Legend at 0x7cc7584e28c0>



In [1]:
```
# Implementing feedforward neural networks with Keras and TensorFlow
# import the necessary packages
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Conv2D,MaxPooling2D,Flatten,Dropout
import matplotlib.pyplot as plt
```

In [6]:
```
# grab the MNIST dataset (if this is your first time using this
# dataset then the 11MB download may take a minute)

print("[INFO] accessing MNIST...")
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.reshape((x_train.shape[0], 28, 28, 1)).astype('float32') / 255
x_test = x_test.reshape((x_test.shape[0], 28, 28, 1)).astype('float32') / 255
```

```
[INFO] accessing MNIST...
```

In [8]:
```
model = Sequential()

# Convolutional layers
model.add(Conv2D(28,kernel_size=(3, 3), input_shape=(28, 28, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())

# Fully connected layers
model.add(Dense(200,activation="relu"))
model.add(Dropout(0.3))
model.add(Dense(10,activation="softmax"))
model.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.
py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a laye
r. When using Sequential models, prefer using an `Input(shape)` object as the fir
st layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape
conv2d (Conv2D)	(None, 26, 26, 28)
max_pooling2d (MaxPooling2D)	(None, 13, 13, 28)
flatten (Flatten)	(None, 4732)
dense (Dense)	(None, 200)
dropout (Dropout)	(None, 200)
dense_1 (Dense)	(None, 10)

Total params: 948,890 (3.62 MB)

Trainable params: 948,890 (3.62 MB)

```
Non-trainable params: 0 (0.00 B)
```

In [9]:
```
# Step 4: Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=
```

In [10]:
```
# Step 5: Train the model
model.fit(x_train, y_train, epochs=2)
```

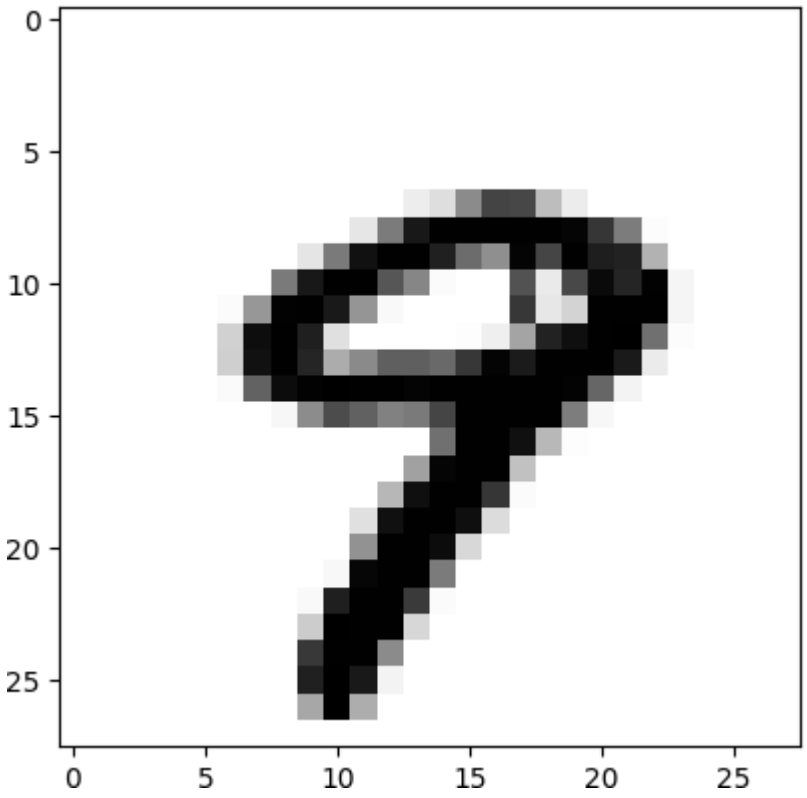
```
Epoch 1/2
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 14s 5ms/step - accuracy: 0.8935 - loss: 0.3502
Epoch 2/2
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 13s 3ms/step - accuracy: 0.9728 - loss: 0.0882
```

Out[10]:
```
<keras.src.callbacks.history.History at 0x7a410db96fb0>
```

In [11]:
```
# Step 6: Evaluate the network
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_accuracy*100:.2f}%')
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.9757 - loss: 0.0757
Test accuracy: 98.07%
```

In [24]:
```
image = x_test[9]
plt.imshow(image,cmap='Greys')
plt.show()
```



In [25]:
```
image=image.reshape(1,28,28,1)
prediction = model.predict(image)
print(np.argmax(prediction))
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 256ms/step
9
```

An autoencoder is a special type of neural network that is trained to copy its input to its output. For example, given an image of a handwritten digit, an autoencoder first encodes the image into a lower dimensional latent representation, then decodes the latent representation back to an image. An autoencoder learns to compress the data while minimizing the reconstruction error.

To learn more about autoencoders, please consider reading chapter 14 from Deep Learning by Ian Goodfellow, Yoshua Bengio, and Aaron Courville.

In [1]:
```
#Import TensorFlow and other libraries
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import tensorflow as tf

from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.model_selection import train_test_split
from tensorflow.keras import layers, losses
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.models import Model
```

Load the dataset

To start, you will train the basic autoencoder using the Fashion MNIST dataset. Each image in this dataset is 28x28 pixels.

In [2]:
```
(x_train, _), (x_test, _) = fashion_mnist.load_data()

x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

print (x_train.shape)
print (x_test.shape)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 ━━━━━━━━━━━━━━━━━━━━ 0s 1us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
(60000, 28, 28)
(10000, 28, 28)
```

First example: Basic autoencoder Define an autoencoder with two Dense layers: an encoder, which compresses the images into a 64 dimensional latent vector, and a decoder, that reconstructs the original image from the latent space.

In [3]:
```
latent_dim = 64
```

```
class Autoencoder(Model):
  def __init__(self, latent_dim):
    super(Autoencoder, self).__init__()
    self.latent_dim = latent_dim
    self.encoder = tf.keras.Sequential([
      layers.Flatten(),
      layers.Dense(latent_dim, activation='relu'),
    ])
    self.decoder = tf.keras.Sequential([
      layers.Dense(784, activation='sigmoid'),
      layers.Reshape((28, 28))
    ])

  def call(self, x):
    encoded = self.encoder(x)
    decoded = self.decoder(encoded)
    return decoded

autoencoder = Autoencoder(latent_dim)
```

In [4]:
```
autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())
```

Train the model using x_train as both the input and the target. The encoder will learn to compress the dataset from 784 dimensions to the latent space, and the decoder will learn to reconstruct the original images. .

In [5]:
```
autoencoder.fit(x_train, x_train,
                epochs=10,
                shuffle=True,
                validation_data=(x_test, x_test))
```

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 4ms/step - loss: 0.0395 - val_loss: 0.0134
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step - loss: 0.0123 - val_loss: 0.0105
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - loss: 0.0102 - val_loss: 0.0099
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - loss: 0.0095 - val_loss: 0.0093
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - loss: 0.0092 - val_loss: 0.0093
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 3ms/step - loss: 0.0090 - val_loss: 0.0091
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - loss: 0.0089 - val_loss: 0.0089
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - loss: 0.0088 - val_loss: 0.0089
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - loss: 0.0088 - val_loss: 0.0088
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - loss: 0.0087 - val_loss: 0.0088
```

Out[5]: <keras.src.callbacks.history.History at 0x7e758e2c7880>

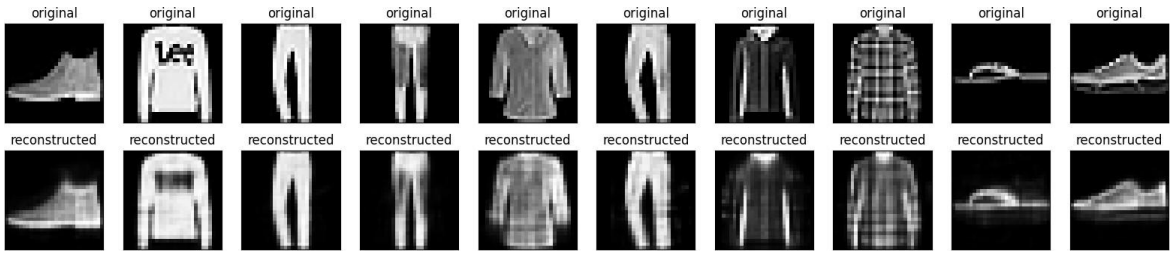
Now that the model is trained, let's test it by encoding and decoding images from the test set.

In [6]:
```
encoded_imgs = autoencoder.encoder(x_test).numpy()

decoded_imgs = autoencoder.decoder(encoded_imgs).numpy()
```

In [7]:
```
n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
  # display original
  ax = plt.subplot(2, n, i + 1)
  plt.imshow(x_test[i])
  plt.title("original")
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)

  # display reconstruction
  ax = plt.subplot(2, n, i + 1 + n)
  plt.imshow(decoded_imgs[i])
  plt.title("reconstructed")
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
plt.show()
```



In [12]:
```
import re
import numpy as np
import string
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline

from subprocess import check_output
from wordcloud import WordCloud, STOPWORDS

stopwords = set(STOPWORDS)
sentences ="""We are about to study the idea of a computational process.
Computational processes are abstract beings that inhabit computers.
As they evolve, processes manipulate other abstract things called data.
The evolution of a process is directed by a pattern of rules
called a program. People create programs to direct processes. In effect,
we conjure the spirits of the computer with our spells."""

wordcloud = WordCloud(
                          background_color='white',
                          stopwords=stopwords,
                          max_words=200,
                          max_font_size=40,
                          random_state=42
                         ).generate(sentences)

fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(10, 10))
axes.imshow(wordcloud)
axes.axis('off')
fig.tight_layout()
```

In [ ]:
```
sentences = """We are about to study the idea of a computational process.
Computational processes are abstract beings that inhabit computers.
As they evolve, processes manipulate other abstract things called data.
The evolution of a process is directed by a pattern of rules
called a program. People create programs to direct processes. In effect,
we conjure the spirits of the computer with our spells."""
```

In [15]:
```
# remove special characters
sentences = re.sub('[^A-Za-z0-9]+', ' ', sentences)

# remove 1 letter words
sentences = re.sub(r'(?:^| )\w(?:$| )', ' ', sentences).strip()

# lower all characters
sentences = sentences.lower()
print(sentences)
```

```
we are about to study the idea of computational process computational processes a
re abstract beings that inhabit computers as they evolve processes manipulate oth
er abstract things called data the evolution of process is directed by pattern of
rules called program people create programs to direct processes in effect we conj
ure the spirits of the computer with our spells
```

In [18]:
```
words = sentences.split()
vocab = set(words)
print(words)
print(vocab)
```

```
['we', 'are', 'about', 'to', 'study', 'the', 'idea', 'of', 'computational', 'proc
ess', 'computational', 'processes', 'are', 'abstract', 'beings', 'that', 'inhabi
t', 'computers', 'as', 'they', 'evolve', 'processes', 'manipulate', 'other', 'abs
tract', 'things', 'called', 'data', 'the', 'evolution', 'of', 'process', 'is', 'd
irected', 'by', 'pattern', 'of', 'rules', 'called', 'program', 'people', 'creat
e', 'programs', 'to', 'direct', 'processes', 'in', 'effect', 'we', 'conjure', 'th
e', 'spirits', 'of', 'the', 'computer', 'with', 'our', 'spells']
{'direct', 'they', 'called', 'rules', 'in', 'effect', 'computer', 'computationa
l', 'about', 'manipulate', 'things', 'pattern', 'directed', 'are', 'other', 'peop
le', 'as', 'inhabit', 'evolve', 'create', 'that', 'study', 'the', 'process', 'abs
tract', 'of', 'idea', 'our', 'data', 'with', 'program', 'programs', 'is', 'by',
'to', 'computers', 'conjure', 'spirits', 'processes', 'evolution', 'we', 'spell
s', 'beings'}
```

In [20]:
```
vocab_size = len(vocab)
embed_dim = 10
context_size = 2
```

In [23]:
```
word_to_ix = {word: i for i, word in enumerate(vocab)}
ix_to_word = {i: word for i, word in enumerate(vocab)}
print(word_to_ix)
print(ix_to_word)
```

```
{'direct': 0, 'they': 1, 'called': 2, 'rules': 3, 'in': 4, 'effect': 5, 'compute
r': 6, 'computational': 7, 'about': 8, 'manipulate': 9, 'things': 10, 'pattern':
11, 'directed': 12, 'are': 13, 'other': 14, 'people': 15, 'as': 16, 'inhabit': 1
7, 'evolve': 18, 'create': 19, 'that': 20, 'study': 21, 'the': 22, 'process': 23,
'abstract': 24, 'of': 25, 'idea': 26, 'our': 27, 'data': 28, 'with': 29, 'progra
m': 30, 'programs': 31, 'is': 32, 'by': 33, 'to': 34, 'computers': 35, 'conjure':
36, 'spirits': 37, 'processes': 38, 'evolution': 39, 'we': 40, 'spells': 41, 'bei
ngs': 42}
{0: 'direct', 1: 'they', 2: 'called', 3: 'rules', 4: 'in', 5: 'effect', 6: 'compu
ter', 7: 'computational', 8: 'about', 9: 'manipulate', 10: 'things', 11: 'patter
n', 12: 'directed', 13: 'are', 14: 'other', 15: 'people', 16: 'as', 17: 'inhabi
t', 18: 'evolve', 19: 'create', 20: 'that', 21: 'study', 22: 'the', 23: 'proces
s', 24: 'abstract', 25: 'of', 26: 'idea', 27: 'our', 28: 'data', 29: 'with', 30:
'program', 31: 'programs', 32: 'is', 33: 'by', 34: 'to', 35: 'computers', 36: 'co
njure', 37: 'spirits', 38: 'processes', 39: 'evolution', 40: 'we', 41: 'spells',
42: 'beings'}
```

In [27]:
```
# data - [(context), target]

data = []
for i in range(2, len(words) - 2):
    context = [words[i - 2], words[i - 1], words[i + 1], words[i + 2]]
    target = words[i]
    data.append((context, target))
print(data[:5])
```

```
[(['we', 'are', 'to', 'study'], 'about'), (['are', 'about', 'study', 'the'], 't
o'), (['about', 'to', 'the', 'idea'], 'study'), (['to', 'study', 'idea', 'of'],
'the'), (['study', 'the', 'of', 'computational'], 'idea')]
```

In [30]:
```
embeddings =  np.random.random_sample((vocab_size,  embed_dim))
print(embeddings)
```

```
[[0.55902317 0.8796336  0.01606508 0.83640464 0.94515038 0.03671365
  0.27138932 0.69865781 0.7621218  0.49927705]
 [0.10055462 0.63714277 0.42372804 0.478108   0.26711547 0.09149576
  0.29168269 0.94036423 0.22413572 0.39162099]
 [0.95464716 0.38335215 0.68628304 0.01724722 0.65466762 0.82630526
  0.14338801 0.47979158 0.55403334 0.59806094]
 [0.84383432 0.23158652 0.49851811 0.52690493 0.19314359 0.03769864
  0.9239182  0.58604157 0.29585361 0.75399642]
 [0.30806521 0.34873435 0.68050268 0.88605163 0.34890095 0.04797554
  0.69039153 0.86210584 0.19452842 0.27652878]
 [0.44921168 0.01018458 0.10001453 0.36017239 0.37986563 0.57077681
  0.36082491 0.15131566 0.79828081 0.21166546]
 [0.10538946 0.86263733 0.97936195 0.80499425 0.47077666 0.15855748
  0.20198191 0.74666776 0.11708653 0.26319007]
 [0.96836749 0.51423675 0.46540649 0.4272708  0.45180645 0.78393261
  0.93397244 0.53718584 0.36399799 0.12210278]
 [0.29130761 0.14499853 0.85154888 0.44465339 0.58324302 0.64895118
  0.13842134 0.52480933 0.69221816 0.15314234]
 [0.89677092 0.08651211 0.63968209 0.17938769 0.92941206 0.96729667
  0.93935048 0.90278976 0.61506383 0.55344748]
 [0.47560007 0.57787429 0.50357385 0.99180738 0.03867795 0.61980775
  0.42743254 0.30866664 0.57509802 0.72280169]
 [0.0292131  0.18008654 0.75532664 0.18913698 0.48467134 0.49147614
  0.99957892 0.1025795  0.31481234 0.59640461]
 [0.34053374 0.88620595 0.32149636 0.75317574 0.90496078 0.16534758
  0.11691434 0.5436065  0.42417629 0.16822331]
 [0.63267794 0.82218887 0.50497123 0.57612382 0.26080854 0.08491206
  0.92220003 0.99517395 0.65428914 0.56916061]
 [0.59867719 0.32500871 0.01006457 0.46699615 0.91691341 0.44061327
  0.69203526 0.19498461 0.27813834 0.48776731]
 [0.03737713 0.76213107 0.29010761 0.30563751 0.68342246 0.64852369
  0.3830537  0.62640629 0.54361764 0.99560582]
 [0.71125134 0.32188125 0.19249123 0.5940112  0.65361697 0.5670915
  0.06405644 0.89447002 0.02501037 0.02569859]
 [0.70682621 0.1697142  0.45092039 0.88107085 0.61191993 0.39235914
  0.92079698 0.62779889 0.16279128 0.81865555]
 [0.5687993  0.22804503 0.93789164 0.07070246 0.35066001 0.53565905
  0.64651151 0.61669068 0.54499238 0.7218042 ]
 [0.39141204 0.95963752 0.08899486 0.57230914 0.15191606 0.07459051
  0.16198143 0.34555402 0.8758638  0.28783637]
 [0.61501984 0.37891144 0.38973929 0.68123779 0.66492461 0.29816377
  0.9266779  0.56303613 0.53299404 0.06964219]
 [0.06740883 0.41160211 0.50818432 0.57372577 0.66085666 0.1742478
  0.20957445 0.52229692 0.60760105 0.05414961]
 [0.35368909 0.35298248 0.96208777 0.18958784 0.65711821 0.33987877
  0.26141294 0.03329625 0.63378331 0.34383058]
 [0.55568049 0.19620255 0.09678858 0.88879468 0.00218402 0.85234013
  0.55104576 0.82430225 0.8379663  0.93048525]
 [0.92481104 0.98222132 0.80940994 0.2548421  0.64036237 0.38058139
  0.95220029 0.3352988  0.54985922 0.64149738]
 [0.49328692 0.50287429 0.03606609 0.88675683 0.53395749 0.45546944
  0.16167272 0.79382407 0.06654294 0.82510425]
 [0.72163701 0.2467374  0.7303794  0.25640706 0.12220016 0.9165005
  0.10952576 0.91633619 0.96842452 0.73890695]
 [0.36586411 0.7515007  0.14722247 0.6406858  0.68766514 0.21646427
  0.14877631 0.72782687 0.14050458 0.74958574]
 [0.13573147 0.91991145 0.13750531 0.88963138 0.34709709 0.76010183
  0.09644531 0.51585206 0.57848192 0.3682413 ]
 [0.35574132 0.06437098 0.01449723 0.01749394 0.99009996 0.87998187
  0.47205441 0.85277301 0.79077788 0.0543162 ]
```

```
[0.61285547 0.71782534 0.25427413  0.09433277 0.45767025 0.03140795
 0.47322179 0.17184041 0.88348221  0.35366897]
[0.69421424 0.23188181 0.76184257  0.61563732 0.54398131 0.25470129
 0.57724233 0.52300325 0.05717392  0.63644609]
[0.52694406 0.4356314  0.45366411  0.09911894 0.59120233 0.37231263
 0.25315644 0.36820299 0.01271292  0.70130125]
[0.89182372 0.98752574 0.81202508  0.64138494 0.05069737 0.21197388
 0.38676834 0.55902303 0.53836614  0.60716916]
[0.60786441 0.92728581 0.69198122  0.61599739 0.74123805 0.35148439
 0.37519341 0.62511948 0.5415566   0.90589371]
[0.16547332 0.98042449 0.78742008  0.17028886 0.54396077 0.14550411
 0.94306519 0.41288371 0.30342499  0.39803867]
[0.46223938 0.29348248 0.04374014  0.40268991 0.06706729 0.39876046
 0.17708012 0.29227249 0.3954279   0.67426925]
[0.91588121 0.08722407 0.48936658  0.96222817 0.25489614 0.23560044
 0.39150502 0.1000599  0.72920947  0.3170044 ]
[0.37038171 0.10456142 0.9623594   0.44572792 0.31929118 0.95818652
 0.70486655 0.28360931 0.87294152  0.95132912]
[0.11802475 0.14982839 0.56022318  0.40463393 0.06156818 0.44766894
 0.63876637 0.35104739 0.24342893  0.51628805]
[0.36886228 0.97610319 0.9001242   0.07279782 0.31235222 0.75657946
 0.20040641 0.12424929 0.56093482  0.53550702]
[0.21426268 0.89639685 0.45833907  0.01409733 0.07841329 0.51662104
 0.30730498 0.29657929 0.86601284  0.71593723]
[0.43256503 0.82074468 0.45736564  0.95366652 0.88362831 0.75057239
 0.84926887 0.02484717 0.56286909 0.85824221]]
```

Reference

https://www.youtube.com/watch?v=AwOIgOwaLl0

Imports

```
import tensorflow_datasets as tfds
import tensorflow as tf
from tensorflow.keras.utils import to_categorical
```

Load Data

Explore more about datset: https://www.tensorflow.org/datasets/catalog/tf_flowers

```
## Loading images and labels
(train_ds, train_labels), (test_ds, test_labels) = tfds.load("tf_flowers",
    split=["train[:70%]", "train[:30%]"], ## Train test split
    batch_size=-1,
    as_supervised=True,   # Include labels
)
```

{"model_id":"ffde15e341f544498356af997cb2caee","version_major":2,"version_minor":0}

```
Dataset tf_flowers downloaded and prepared to
/root/tensorflow_datasets/tf_flowers/3.0.1. Subsequent calls will reuse this
data.
```

Image Preprocessing

```
## check existing image size
train_ds[0].shape
```

```
## Resizing images
train_ds = tf.image.resize(train_ds, (150, 150))
test_ds = tf.image.resize(test_ds, (150, 150))
train_ds[0].shape
```

```
## Transforming labels to correct format
train_labels = to_categorical(train_labels, num_classes=5)
test_labels = to_categorical(test_labels, num_classes=5)
```

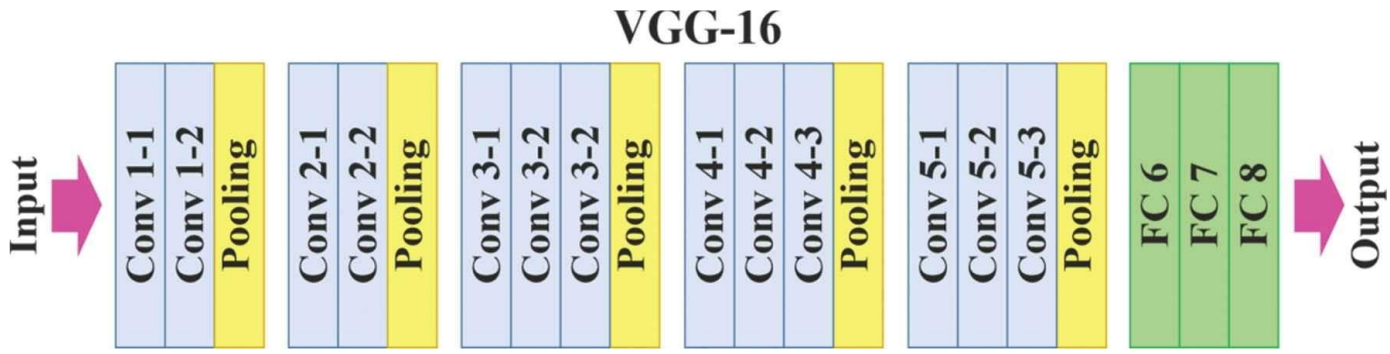
Use Pretrained VGG16 Image Classification model

Load a pre-trained CNN model trained on a large dataset

```
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input

## Loading VGG16 model
base_model = VGG16(weights="imagenet", include_top=False,
input_shape=train_ds[0].shape)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 ——————————————————5s 0us/step
```




```
## We will not train base model i.e. Freeze Parameters in model's lower
convolutional layers
base_model.trainable = False

## Preprocessing input
train_ds = preprocess_input(train_ds)
test_ds = preprocess_input(test_ds)

## model details
base_model.summary()
```

```
Model: "vgg16"
```

```
┌─────────────────────────────────────┬─────────────────────────────┬──────────
──────┐
| Layer (type)                        | Output Shape                |
Param # |
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
| input_layer (InputLayer)            | (None, 150, 150, 3)         |
0 |
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
| block1_conv1 (Conv2D)               | (None, 150, 150, 64)        |
1,792 |
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
| block1_conv2 (Conv2D)               | (None, 150, 150, 64)        |
36,928 |
├─────────────────────────────────────┼─────────────────────────────┼──────────
```

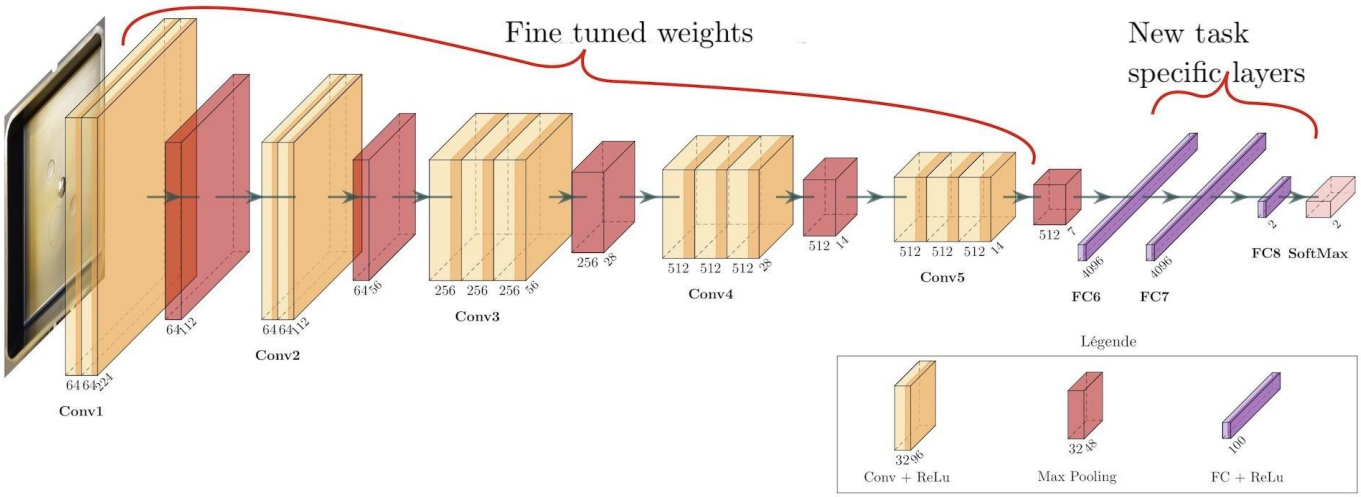
```
──────────┤
 block1_pool (MaxPooling2D)              (None, 75, 75, 64)            
0  
├──────────────────────────────────────┼──────────────────────────────┼──────
──────────┤
 block2_conv1  (Conv2D)                  (None, 75, 75, 128)           
73,856  
├──────────────────────────────────────┼──────────────────────────────┼──────
──────────┤
 block2_conv2  (Conv2D)                  (None, 75, 75, 128)           
147,584  
├──────────────────────────────────────┼──────────────────────────────┼──────
──────────┤
 block2_pool (MaxPooling2D)              (None, 37, 37, 128)           
0  
├──────────────────────────────────────┼──────────────────────────────┼──────
──────────┤
 block3_conv1  (Conv2D)                  (None, 37, 37, 256)           
295,168  
├──────────────────────────────────────┼──────────────────────────────┼──────
──────────┤
 block3_conv2  (Conv2D)                  (None, 37, 37, 256)           
590,080  
├──────────────────────────────────────┼──────────────────────────────┼──────
──────────┤
 block3_conv3  (Conv2D)                  (None, 37, 37, 256)           
590,080  
├──────────────────────────────────────┼──────────────────────────────┼──────
──────────┤
 block3_pool (MaxPooling2D)              (None, 18, 18, 256)           
0  
├──────────────────────────────────────┼──────────────────────────────┼──────
──────────┤
 block4_conv1  (Conv2D)                  (None, 18, 18, 512)           
1,180,160  
├──────────────────────────────────────┼──────────────────────────────┼──────
──────────┤
 block4_conv2  (Conv2D)                  (None, 18, 18, 512)           
2,359,808  
├──────────────────────────────────────┼──────────────────────────────┼──────
──────────┤
 block4_conv3  (Conv2D)                  (None, 18, 18, 512)           
2,359,808  
├──────────────────────────────────────┼──────────────────────────────┼──────
──────────┤
 block4_pool (MaxPooling2D)              (None, 9, 9, 512)             
0  
├──────────────────────────────────────┼──────────────────────────────┼──────
──────────┤
 block5_conv1  (Conv2D)                  (None, 9, 9, 512)             
2,359,808  
├──────────────────────────────────────┼──────────────────────────────┼──────
──────────┤
 block5_conv2  (Conv2D)                  (None, 9, 9, 512)             
2,359,808  
├──────────────────────────────────────┼──────────────────────────────┼──────
──────────┤
 block5_conv3  (Conv2D)                  (None, 9, 9, 512)             
```

```
2,359,808  |
 ├─────────────────────────────────┼──────────────────────────────┼──────
──────────┤
 | block5_pool (MaxPooling2D)              | (None, 4, 4, 512)            |
0  |
 └─────────────────────────────────┴──────────────────────────────┴──────
──────────┘

 Total params: 14,714,688 (56.13 MB)

 Trainable params: 0 (0.00 B)

 Non-trainable params: 14,714,688 (56.13 MB)
```



Add custom classifier with two dense layers of trainable parameters to model

```
#add our layers on top of this model
from tensorflow.keras import layers, models

flatten_layer = layers.Flatten()
dense_layer_1 = layers.Dense(50, activation='relu')
dense_layer_2 = layers.Dense(20, activation='relu')
prediction_layer = layers.Dense(5, activation='softmax')


model = models.Sequential([
    base_model,
    flatten_layer,
    dense_layer_1,
    dense_layer_2,
    prediction_layer
])
```

Train classifier layers on training data available for task

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

history=model.fit(train_ds, train_labels, epochs=10, validation_split=0.2,
batch_size=32)
```

```
Epoch 1/5
65/65 ━━━━━━━━━━━━━━━━━━━━ 33s 274ms/step - accuracy: 0.3618 - loss: 2.6283 -
val_accuracy: 0.5156 - val_loss: 1.1623
Epoch 2/5
65/65 ━━━━━━━━━━━━━━━━━━━━ 21s 174ms/step - accuracy: 0.5954 - loss: 1.0192 -
val_accuracy: 0.6284 - val_loss: 1.0488
Epoch 3/5
65/65 ━━━━━━━━━━━━━━━━━━━━ 11s 168ms/step - accuracy: 0.7250 - loss: 0.7869 -
val_accuracy: 0.6634 - val_loss: 1.0971
Epoch 4/5
65/65 ━━━━━━━━━━━━━━━━━━━━ 21s 179ms/step - accuracy: 0.7849 - loss: 0.5597 -
val_accuracy: 0.6518 - val_loss: 0.9481
Epoch 5/5
65/65 ━━━━━━━━━━━━━━━━━━━━ 8s 127ms/step - accuracy: 0.8399 - loss: 0.4183 -
val_accuracy: 0.7043 - val_loss: 0.9855
```

```python
los,accurac=model.evaluate(test_ds,test_labels)
print("Loss: ",los,"Accuracy: ", accurac)
```

```
35/35 ━━━━━━━━━━━━━━━━━━━━ 13s 385ms/step - accuracy: 0.8811 - loss: 0.3352
Loss:  0.3205247223377228 Accuracy:  0.8864668607711792
```

```python
import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'])
plt.title('ACCURACY')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train'],loc='upper left')
plt.show()
```