Report: Movie Review Binary Classification

This paper describes the creation of a Keras neural network for binary categorization of movie reviews from the IMDB movie reviews dataset. The neural network's aim is to identify whether an evaluation is favorable or negative.

Problem Statment:

The objective of this project is to use Keras to build a deep neural network to predict whether a review will be positive or negative.

Dataset:

The IMDB dataset contains 50,000 highly polarized movie reviews that have been split into two equal groups of 25,000 reviews each for training and testing. There are an equal amount of favorable and negative evaluations in each set. Keras includes this dataset, which contains reviews and the labels that go with them, with 0 indicating a negative review and 1 reflecting a good review. The evaluations take the shape of a series of words that have been pre-processed into a series of numbers, each integer signifying a different word from the dictionary.

Methodology:

To create the neural network, we used Python's Keras framework. First, the dataset was split into training and test groups. The data was then loaded, with only the 10,000 most commonly appearing terms retained. The evaluations were then decoded to their original text using a lexicon mapping from word to integer value.

A sequential model was then developed, consisting of an embedding layer that takes the integer-encoded vocabulary and translates each word to a feature vector of a given size. This was followed by a dense layer with 16 hidden units and the ReLU activation function, which was then linked to an output layer with a sigmoid activation function to generate a chance between 0 and 1. The model was built with binary cross-entropy as the loss function, the RMSprop optimizer, and accuracy as the assessment measure.

Loading the IMDB dataset

```
# importing libraries

import matplotlib
import matplotlib.pyplot as plt
import numpy as np
from keras import models
from keras import layers
```

Dataset is divided into train data and test data

Data is loaded and 10,000 most frequently occuring words are kept

```
from tensorflow.keras.datasets import imdb
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(
    num_words=10000)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 [==============================] - 0s 0us/step
```

Printing training data first review

```
train_data[0]
```

```
[1,
 14,
 22,
 16,
 43,
 530,
 973,
 1622,
 1385,
 65,
 458,
 4468,
 66,
 3941,
 4,
 173,
 36,
 256,
 5,
```

```
      25,
      100,
      43,
      838,
      112,
      50,
      670,
      2,
      9,
      35,
      480,
      284,
      5,
      150,
      4,
      172,
      112,
      167,
      2,
      336,
      385,
      39,
      4,
      172,
      4536,
      1111,
      17,
      546,
      38,
      13,
      447,
      4,
      192,
      50,
      16,
      6,
      147,
      2025,
      19,
```

```
# Check the first label

train_labels[0]
```

```
     1
```

```
# 10000 frequent words are taken into account and  word index should not be exceeded to the frequent word count.

# Finding the max of all the max indexes
max([max(sequence) for sequence in train_data])
```

```
     9999
```

Decoding movie reviews back to text

```
#Loading the mappings from word to integer index and reverse the word index to integer  and then decode the review by mapping inte

# step 1: Load the mappings of the dictionary from the word to integer index
word_index = imdb.get_word_index()

# step 2: reverse word index to integer mapping
reverse_word_index = dict(
    [(value, key) for (key, value) in word_index.items()])

# step 3: Decode the review, mapping integer to words
decoded_review = " ".join(
    [reverse_word_index.get(i - 3, "?") for i in train_data[0]])
decoded_review
```

```
    Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb_word_index.json
len(reverse_word_index)

    88584

    t was released for ? and would recommend it to everyone to watch and the fly fishing was amazing really cried at the end it w
```

Preparing the data:

- Our deep convolutional neural network cannot be provided a list of numbers. We'll have to transform them into tensors.
- To prepare our data, we'll use One-hot Encoding to transform our lists into vectors of 0s and 1s. Each of our sequences would be blown up into 10,000-dimensional vectors with 1 at all positions connecting all numbers in the sequence.
- This vector will contain the element 0 for all values that are not in integer order.Simply put, each evaluation will be symbolized by a 10,000-dimensional vector.Each index correlates to a particular phrase.
- Every index with the value 1 indicates a term in the review that is marked by its integer equivalent. Every number that begins with 0 is a name.

```
import numpy as np
def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        for j in sequence:
            results[i, j] = 1
    return results

# Vectorizing the training and test data
x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)


x_train[0]

    array([0., 1., 1., ..., 0., 0., 0.])


x_train.shape


    (25000, 10000)


y_train = np.asarray(train_labels).astype("float32")
y_test = np.asarray(test_labels).astype("float32")
```

Making the model:

Our initial dataset was made up of vectors that needed to be transformed to encoder labels (0s and 1s). This is one of the most fundamental configurations, and a basic stack of completely linked Dense layers with relu activation works well.

Hidden Layers:

- In this network, we'll use concealed levels. As a result, we'll categorize our levels.
- Dense(16,'relu' activation) The parameter given to each Dense layer specifies the number of concealed units of a layer (16).
- The output of a Dense layer with relu activation is produced after a sequence of tensor processes. The following is how this procedure is carried out:
- relu(dot(W, input) Plus b) output W is the weight matrix, and b is the skew (tensor).
- If there are 16 concealed units, the matrix W will have the form (_inputDimension, 16). The dimension of the input vector in this situation is 10,000, and the construction of the Weight matrix is (10000, 16). If this network were depicted as a graph, it would have 16 nodes.

Architecture of the Model

We will be utilizing the for our example.

- Two intermediary levels, each with 16 hidden layers, employ the relu activation function, which is used to zero out negative numbers.
- The output layer, which employs sigmoid activation, will be the third layer

```
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
    layers.Dense(16, activation="relu"),
```

```
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
```

Compiling the model:

We will look at the optimizer, loss function, and data during the compilation phase.

The following methods will be used in this case.

- The binary crossentropy loss function is employed in binary categorization. rmsprop is the algorithm used.
- Accuracy is used to assess success.
- Because keras includes all of the rmsprop, binary crossentropy, and accuracy functions, the above methods can be used to build the model.

model.compile(rmsprop, loss="binary crossentropy", metrics="accuracy"])

```
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
```

Setting aside a validation set

We'll save some of our training data to test the model's accuracy as it advances. A validation set enables businesses to track our model's development through epochs during training on previously unknown data.

- We can fine-tune the model's training settings using validation stages.
- To avoid data overfitting and underfitting, use the fit tool.

```
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

Training the model

For the first 20 epochs, we'll train our models in 512-sample mini-batches. Our confirmation collection will also be used by the fit technique.

When invoked, the fit method will return a History object. This object has a member history that contains all training process information, including the values of visible or tracked quantities as the epochs advance. This object will be saved so that we can better determine how to fine-tune the training process.

```
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/20
30/30 [==============================] - 4s 74ms/step - loss: 0.5250 - accuracy: 0.7859 - val_loss: 0.4109 - val_accuracy: 0.
Epoch 2/20
30/30 [==============================] - 1s 40ms/step - loss: 0.3291 - accuracy: 0.8950 - val_loss: 0.3196 - val_accuracy: 0.
Epoch 3/20
30/30 [==============================] - 1s 40ms/step - loss: 0.2505 - accuracy: 0.9179 - val_loss: 0.2872 - val_accuracy: 0.
Epoch 4/20
30/30 [==============================] - 1s 35ms/step - loss: 0.2045 - accuracy: 0.9321 - val_loss: 0.2773 - val_accuracy: 0.
Epoch 5/20
30/30 [==============================] - 1s 38ms/step - loss: 0.1704 - accuracy: 0.9442 - val_loss: 0.2866 - val_accuracy: 0.
Epoch 6/20
30/30 [==============================] - 1s 38ms/step - loss: 0.1455 - accuracy: 0.9541 - val_loss: 0.2867 - val_accuracy: 0.
Epoch 7/20
30/30 [==============================] - 1s 39ms/step - loss: 0.1271 - accuracy: 0.9605 - val_loss: 0.2910 - val_accuracy: 0.
Epoch 8/20
30/30 [==============================] - 1s 37ms/step - loss: 0.1081 - accuracy: 0.9689 - val_loss: 0.3146 - val_accuracy: 0.
Epoch 9/20
30/30 [==============================] - 2s 52ms/step - loss: 0.0966 - accuracy: 0.9715 - val_loss: 0.3561 - val_accuracy: 0.
Epoch 10/20
30/30 [==============================] - 2s 65ms/step - loss: 0.0828 - accuracy: 0.9787 - val_loss: 0.3327 - val_accuracy: 0.
Epoch 11/20
30/30 [==============================] - 1s 40ms/step - loss: 0.0723 - accuracy: 0.9801 - val_loss: 0.3506 - val_accuracy: 0.
Epoch 12/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0642 - accuracy: 0.9836 - val_loss: 0.3645 - val_accuracy: 0.
Epoch 13/20
```

```
30/30 [==============================] - 1s 40ms/step - loss: 0.0546 - accuracy: 0.9868 - val_loss: 0.3832 - val_accuracy: 0.
Epoch 14/20
30/30 [==============================] - 1s 39ms/step - loss: 0.0482 - accuracy: 0.9885 - val_loss: 0.4045 - val_accuracy: 0.
Epoch 15/20
30/30 [==============================] - 1s 39ms/step - loss: 0.0408 - accuracy: 0.9918 - val_loss: 0.4564 - val_accuracy: 0.
Epoch 16/20
30/30 [==============================] - 1s 40ms/step - loss: 0.0361 - accuracy: 0.9929 - val_loss: 0.4483 - val_accuracy: 0.
Epoch 17/20
30/30 [==============================] - 1s 40ms/step - loss: 0.0298 - accuracy: 0.9949 - val_loss: 0.4687 - val_accuracy: 0.
Epoch 18/20
30/30 [==============================] - 1s 39ms/step - loss: 0.0256 - accuracy: 0.9961 - val_loss: 0.4942 - val_accuracy: 0.
Epoch 19/20
30/30 [==============================] - 1s 47ms/step - loss: 0.0215 - accuracy: 0.9969 - val_loss: 0.5254 - val_accuracy: 0.
Epoch 20/20
30/30 [==============================] - 2s 54ms/step - loss: 0.0214 - accuracy: 0.9967 - val_loss: 0.5325 - val_accuracy: 0.
```

- By the conclusion of the training, we had achieved a training accuracy of 99.99% and a confirmation accuracy of 86.80%.
- The network's performance measurements are monitored and saved in the history object.
- The match function returns a history object. This entity has a dictionary with four items as a property.

```
history_dict = history.history
history_dict.keys()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

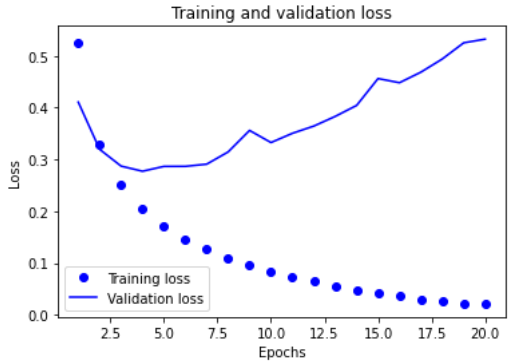
history_dict consists of

- Training loss
- Training Accuracy
- Validation Loss
- Validation Accuracy

We're using Matplotlib to compare the loss and precision of Training and Validation.

Plotting the training and validation loss

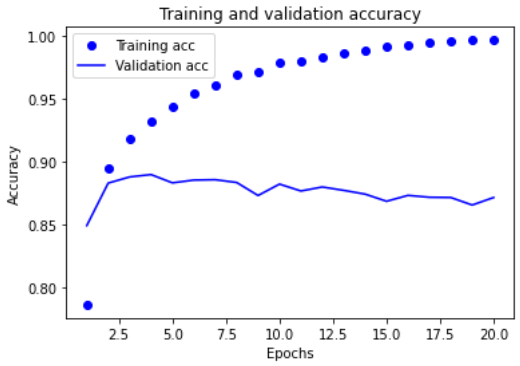
```
# losses
import matplotlib.pyplot as plt
history_dict = history.history
loss_values = history_dict["loss"]
val_loss_values = history_dict["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", label="Training loss")
plt.plot(epochs, val_loss_values, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



Confirmation loss began to increase after the third era. As a result, the remodeling is carried out using the third and fourth epochs.

Plotting the training and validation accuracy

```
# accuracy
plt.clf()
acc = history_dict["accuracy"]
val_acc = history_dict["val_accuracy"]
plt.plot(epochs, acc, "bo", label="Training acc")
plt.plot(epochs, val_acc, "b", label="Validation acc")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



The graphs above show that the lowest validation loss and optimum validation accuracy occur at 3 to 5 epochs. Then we notice two patterns.

- Validation loss rises, while training loss falls.
- The certification accuracy falls while the training accuracy rises.

The preceding consequences imply that while the model improves at classifying the training data, it consistently makes worse forecasts when it meets new and unknown data, indicating overfitting. After the fifth epoch, the algorithm starts to match the training data too closely.

To resolve overfitting, the epoch count is decreased to between 3 and 5 epochs. The epochs may differ based on the machine and the character of randomly allocated weights.

Retraining our model

```
model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])

model.fit(x_train, y_train, epochs=3, batch_size=512)
results = model.evaluate(x_test, y_test)
```

```
Epoch 1/3
49/49 [==============================] - 2s 37ms/step - loss: 0.4683 - accuracy: 0.8092
Epoch 2/3
49/49 [==============================] - 3s 52ms/step - loss: 0.2690 - accuracy: 0.9056
Epoch 3/3
49/49 [==============================] - 2s 41ms/step - loss: 0.2129 - accuracy: 0.9227
782/782 [==============================] - 3s 3ms/step - loss: 0.2819 - accuracy: 0.8885
```

Result

The model was trained for 20 epochs and obtained a test accuracy of 88.72%. To illustrate the findings, the model's training and validation loss and accuracy were plotted on graphs. It was discovered that the training and confirmation precision grew concurrently with each epoch, while the loss decreased. Furthermore, the training accuracy was found to be reliably greater than the validation accuracy, indicating that the model is slightly overfitting.

```
results
```

```
[0.28187671303749084, 0.8885200023651123]
```

1. You used two hidden layers. Try using one or three hidden layers, and see how doing so affects validation and test accuracy.

In this report, we will examine two neural networks with various designs to see how the number of hidden layers affects model accuracy.

The first neural network has only one hidden layer with 16 nodes and uses the ReLU activation function. The output layer is composed of only one node and employs the sigmoid activation function. RMSprop is the algorithm used, and binary cross-entropy is the loss function. The group size has been fixed to 512, and the number of epochs to 20.

The second neural network, like the first, has three hidden layers with 16 nodes each and employs the ReLU activation function. The output layer is configured similarly to the first neural network. RMSprop is the algorithm used, and binary cross-entropy is the loss function. The group size has been fixed to 512, and the number of epochs to 20.

```
# Three hidden layers and one relu activation function model
model_13 = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])

# one hidden layer and one relu activation function model
model_11 = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])


#RMSProp and binary cross entropy for both models
model_13.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])

model_11.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])


# Fitting model with 20 epochs and 512 batch size
history_13 = model_13.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))

history_11 = model_11.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/20
30/30 [==============================] - 3s 61ms/step - loss: 0.5332 - accuracy: 0.7586 - val_loss: 0.4323 - val_accuracy: 0.
Epoch 2/20
30/30 [==============================] - 1s 38ms/step - loss: 0.3073 - accuracy: 0.8925 - val_loss: 0.3243 - val_accuracy: 0.
Epoch 3/20
30/30 [==============================] - 1s 41ms/step - loss: 0.2260 - accuracy: 0.9227 - val_loss: 0.3023 - val_accuracy: 0.
Epoch 4/20
30/30 [==============================] - 1s 40ms/step - loss: 0.1780 - accuracy: 0.9397 - val_loss: 0.2784 - val_accuracy: 0.
Epoch 5/20
30/30 [==============================] - 2s 61ms/step - loss: 0.1513 - accuracy: 0.9478 - val_loss: 0.2845 - val_accuracy: 0.
Epoch 6/20
30/30 [==============================] - 2s 53ms/step - loss: 0.1244 - accuracy: 0.9579 - val_loss: 0.3343 - val_accuracy: 0.
Epoch 7/20
30/30 [==============================] - 1s 39ms/step - loss: 0.1037 - accuracy: 0.9667 - val_loss: 0.3283 - val_accuracy: 0.
Epoch 8/20
30/30 [==============================] - 1s 39ms/step - loss: 0.0867 - accuracy: 0.9732 - val_loss: 0.3578 - val_accuracy: 0.
Epoch 9/20
30/30 [==============================] - 1s 40ms/step - loss: 0.0720 - accuracy: 0.9787 - val_loss: 0.3632 - val_accuracy: 0.
Epoch 10/20
30/30 [==============================] - 1s 37ms/step - loss: 0.0574 - accuracy: 0.9846 - val_loss: 0.3893 - val_accuracy: 0.
Epoch 11/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0532 - accuracy: 0.9843 - val_loss: 0.4140 - val_accuracy: 0.
Epoch 12/20
30/30 [==============================] - 1s 41ms/step - loss: 0.0434 - accuracy: 0.9883 - val_loss: 0.4418 - val_accuracy: 0.
Epoch 13/20
```

```
30/30 [==============================] - 1s 38ms/step - loss: 0.0374 - accuracy: 0.9898 - val_loss: 0.4710 - val_accuracy: 0.
Epoch 14/20
30/30 [==============================] - 1s 40ms/step - loss: 0.0333 - accuracy: 0.9897 - val_loss: 0.4944 - val_accuracy: 0.
Epoch 15/20
30/30 [==============================] - 1s 50ms/step - loss: 0.0326 - accuracy: 0.9897 - val_loss: 0.5170 - val_accuracy: 0.
Epoch 16/20
30/30 [==============================] - 2s 66ms/step - loss: 0.0123 - accuracy: 0.9992 - val_loss: 0.6282 - val_accuracy: 0.
Epoch 17/20
30/30 [==============================] - 1s 39ms/step - loss: 0.0222 - accuracy: 0.9945 - val_loss: 0.6365 - val_accuracy: 0.
Epoch 18/20
30/30 [==============================] - 1s 40ms/step - loss: 0.0240 - accuracy: 0.9937 - val_loss: 0.5933 - val_accuracy: 0.
Epoch 19/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0080 - accuracy: 0.9993 - val_loss: 0.7161 - val_accuracy: 0.
Epoch 20/20
30/30 [==============================] - 1s 37ms/step - loss: 0.0171 - accuracy: 0.9947 - val_loss: 0.6477 - val_accuracy: 0.
Epoch 1/20
30/30 [==============================] - 3s 73ms/step - loss: 0.5078 - accuracy: 0.7894 - val_loss: 0.3988 - val_accuracy: 0.
Epoch 2/20
30/30 [==============================] - 2s 55ms/step - loss: 0.3349 - accuracy: 0.8948 - val_loss: 0.3395 - val_accuracy: 0.
Epoch 3/20
30/30 [==============================] - 1s 37ms/step - loss: 0.2652 - accuracy: 0.9154 - val_loss: 0.2964 - val_accuracy: 0.
Epoch 4/20
30/30 [==============================] - 1s 40ms/step - loss: 0.2239 - accuracy: 0.9282 - val_loss: 0.2950 - val_accuracy: 0.
Epoch 5/20
30/30 [==============================] - 1s 40ms/step - loss: 0.1944 - accuracy: 0.9369 - val_loss: 0.2751 - val_accuracy: 0.
Epoch 6/20
30/30 [==============================] - 1s 40ms/step - loss: 0.1733 - accuracy: 0.9453 - val_loss: 0.2773 - val_accuracy: 0.
Epoch 7/20
30/30 [==============================] - 1s 39ms/step - loss: 0.1557 - accuracy: 0.9524 - val_loss: 0.2802 - val_accuracy: 0.
Epoch 8/20
30/30 [==============================] - 1s 38ms/step - loss: 0.1416 - accuracy: 0.9564 - val_loss: 0.2786 - val_accuracy: 0.
Epoch 9/20
30/30 [==============================] - 1s 41ms/step - loss: 0.1289 - accuracy: 0.9625 - val_loss: 0.2835 - val_accuracy: 0.
```

Plotting training and validation loss

```
historyp_13 = history_13.history
historyp_13.keys()
```

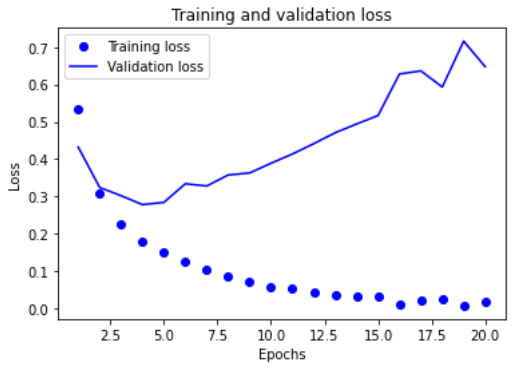
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
historyp_11 = history_11.history
historyp_11.keys()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

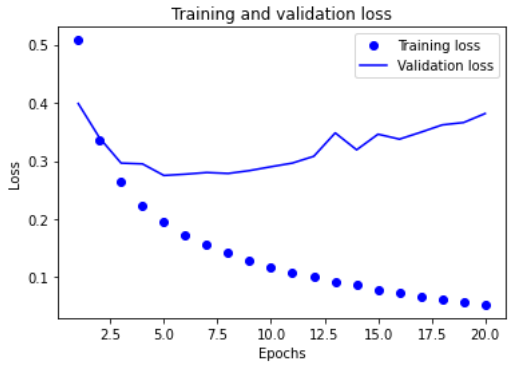
Plotting training and testing loss

```
# Losses
historyp_13 = history_13.history
loss_values3 = historyp_13["loss"]
val_loss_values3 = historyp_13["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values3, "bo", label="Training loss")
plt.plot(epochs, val_loss_values3, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



Here, the minimum validation loss is observed at 5th epoch

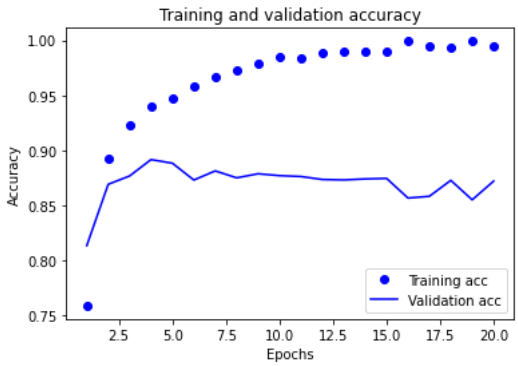
```
#Losses
historyp_11 = history_11.history
loss_values1 = historyp_11["loss"]
val_loss_values1 = historyp_11["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values1, "bo", label="Training loss")
plt.plot(epochs, val_loss_values1, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



Here, the minimum validation loss is observed at 5th epoch

Plotting training and testing accuracy

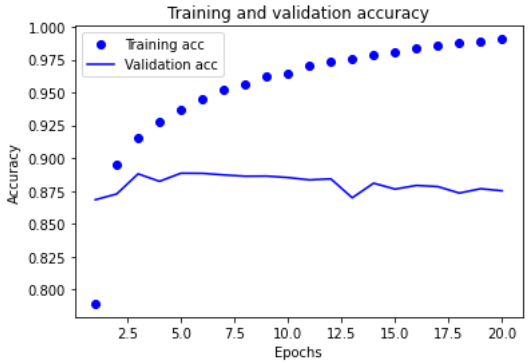
```
#Accuracy
plt.clf()
acc3 = historyp_13["accuracy"]
val_acc3 = historyp_13["val_accuracy"]
plt.plot(epochs, acc3, "bo", label="Training acc")
plt.plot(epochs, val_acc3, "b", label="Validation acc")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



Here, the maximum validation accuracy is observed at 5th epoch

```
#Accuracy
plt.clf()
acc1 = historyp_11["accuracy"]
val_acc1 = historyp_11["val_accuracy"]
plt.plot(epochs, acc1, "bo", label="Training acc")
plt.plot(epochs, val_acc1, "b", label="Validation acc")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
```

```
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



Double-click (or enter) to edit

Here, the maximum vallidation accuracy is observed at 5th epoch.

Results:

Both models are trained on the same dataset, and their performance is evaluated on the validation set, which is different from the training set. The following table shows the accuracy of both neural networks on the validation and test datasets.

Model

One Validation Set : 0.8868 Accuracy on Test Set : 0.8744

Three Validation Set : 0.8824 Accuracy on Test Set : 0.8716

From the table above, we can see that the neural network with one hidden layer performed better than the neural network with three hidden layers. The difference in accuracy is relatively small, but it is consistent on both the validation and test datasets.

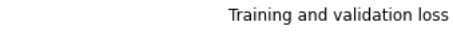
2. Try using layers with more hidden units or fewer hidden units: 32 units, 64 units, and so on.

```
#2 hidden layers and 32 and 64 nodes model
model_2 = keras.Sequential([
    layers.Dense(32, activation="relu"),
    layers.Dense(64, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])


#RMSProp and binary cross entropy
model_2.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])


# Fitting model with 20 epochs and 512 batch size
history2 = model_2.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/20
30/30 [==============================] - 4s 92ms/step - loss: 0.5350 - accuracy: 0.7552 - val_loss: 0.3825 - val_accuracy: 0.
Epoch 2/20
30/30 [==============================] - 1s 45ms/step - loss: 0.3096 - accuracy: 0.8905 - val_loss: 0.2935 - val_accuracy: 0.
Epoch 3/20
30/30 [==============================] - 2s 57ms/step - loss: 0.2290 - accuracy: 0.9164 - val_loss: 0.2756 - val_accuracy: 0.
Epoch 4/20
30/30 [==============================] - 1s 45ms/step - loss: 0.1822 - accuracy: 0.9356 - val_loss: 0.3438 - val_accuracy: 0.
Epoch 5/20
30/30 [==============================] - 2s 57ms/step - loss: 0.1500 - accuracy: 0.9473 - val_loss: 0.2923 - val_accuracy: 0.
Epoch 6/20
30/30 [==============================] - 1s 46ms/step - loss: 0.1266 - accuracy: 0.9568 - val_loss: 0.3869 - val_accuracy: 0.
Epoch 7/20
30/30 [==============================] - 1s 45ms/step - loss: 0.1029 - accuracy: 0.9661 - val_loss: 0.3420 - val_accuracy: 0.
Epoch 8/20
```

```
30/30 [==============================] - 2s 74ms/step - loss: 0.0838 - accuracy: 0.9737 - val_loss: 0.3350 - val_accuracy: 0.
Epoch 9/20
30/30 [==============================] - 2s 58ms/step - loss: 0.0639 - accuracy: 0.9819 - val_loss: 0.4027 - val_accuracy: 0.
Epoch 10/20
30/30 [==============================] - 1s 45ms/step - loss: 0.0639 - accuracy: 0.9805 - val_loss: 0.3853 - val_accuracy: 0.
Epoch 11/20
30/30 [==============================] - 1s 44ms/step - loss: 0.0455 - accuracy: 0.9870 - val_loss: 0.4083 - val_accuracy: 0.
Epoch 12/20
30/30 [==============================] - 1s 44ms/step - loss: 0.0413 - accuracy: 0.9883 - val_loss: 0.4294 - val_accuracy: 0.
Epoch 13/20
30/30 [==============================] - 1s 47ms/step - loss: 0.0175 - accuracy: 0.9985 - val_loss: 0.4739 - val_accuracy: 0.
Epoch 14/20
30/30 [==============================] - 1s 44ms/step - loss: 0.0324 - accuracy: 0.9899 - val_loss: 0.4865 - val_accuracy: 0.
Epoch 15/20
30/30 [==============================] - 1s 45ms/step - loss: 0.0210 - accuracy: 0.9945 - val_loss: 0.5086 - val_accuracy: 0.
Epoch 16/20
30/30 [==============================] - 2s 56ms/step - loss: 0.0079 - accuracy: 0.9997 - val_loss: 0.5975 - val_accuracy: 0.
Epoch 17/20
30/30 [==============================] - 2s 71ms/step - loss: 0.0248 - accuracy: 0.9926 - val_loss: 0.5640 - val_accuracy: 0.
Epoch 18/20
30/30 [==============================] - 1s 48ms/step - loss: 0.0181 - accuracy: 0.9946 - val_loss: 0.5730 - val_accuracy: 0.
Epoch 19/20
30/30 [==============================] - 1s 45ms/step - loss: 0.0039 - accuracy: 0.9999 - val_loss: 0.5982 - val_accuracy: 0.
Epoch 20/20
30/30 [==============================] - 1s 45ms/step - loss: 0.0031 - accuracy: 0.9999 - val_loss: 0.8019 - val_accuracy: 0.
```

```
historyp2 = history2.history
loss_values = historyp2["loss"]
val_loss_values = historyp2["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", label="Training loss")
plt.plot(epochs, val_loss_values, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()

plt.clf()
acc = historyp2["accuracy"]
val_acc = historyp2["val_accuracy"]
plt.plot(epochs, acc, "bo", label="Training acc")
plt.plot(epochs, val_acc, "b", label="Validation acc")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



Double-click (or enter) to edit

The minimum validation loss is observed at 3rd epoch and maximum validation accuracy is observed at 3rd and 4th epochs.

3. Try using the mse loss function instead of binary_crossentropy.

```
model_3 = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
```

```
#RMSProp and mse loss function
model_3.compile(optimizer="rmsprop",
              loss="mse",
              metrics=["accuracy"])
```

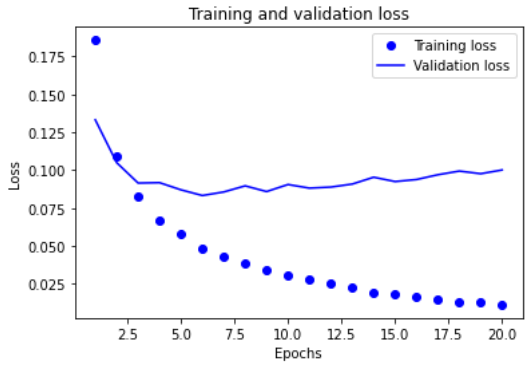
```
# Fitting model with 20 epochs and 512 batch size
history3 = model_3.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/20
30/30 [==============================] - 2s 59ms/step - loss: 0.1857 - accuracy: 0.7639 - val_loss: 0.1331 - val_accuracy: 0.
Epoch 2/20
30/30 [==============================] - 1s 38ms/step - loss: 0.1091 - accuracy: 0.8842 - val_loss: 0.1049 - val_accuracy: 0.
Epoch 3/20
30/30 [==============================] - 1s 39ms/step - loss: 0.0826 - accuracy: 0.9055 - val_loss: 0.0914 - val_accuracy: 0.
Epoch 4/20
30/30 [==============================] - 1s 40ms/step - loss: 0.0670 - accuracy: 0.9255 - val_loss: 0.0917 - val_accuracy: 0.
Epoch 5/20
30/30 [==============================] - 2s 61ms/step - loss: 0.0580 - accuracy: 0.9335 - val_loss: 0.0870 - val_accuracy: 0.
Epoch 6/20
30/30 [==============================] - 2s 50ms/step - loss: 0.0486 - accuracy: 0.9467 - val_loss: 0.0832 - val_accuracy: 0.
Epoch 7/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0428 - accuracy: 0.9557 - val_loss: 0.0856 - val_accuracy: 0.
Epoch 8/20
30/30 [==============================] - 1s 39ms/step - loss: 0.0387 - accuracy: 0.9589 - val_loss: 0.0896 - val_accuracy: 0.
Epoch 9/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0341 - accuracy: 0.9666 - val_loss: 0.0858 - val_accuracy: 0.
Epoch 10/20
30/30 [==============================] - 1s 37ms/step - loss: 0.0309 - accuracy: 0.9688 - val_loss: 0.0905 - val_accuracy: 0.
Epoch 11/20
30/30 [==============================] - 1s 37ms/step - loss: 0.0280 - accuracy: 0.9734 - val_loss: 0.0881 - val_accuracy: 0.
Epoch 12/20
30/30 [==============================] - 1s 40ms/step - loss: 0.0250 - accuracy: 0.9769 - val_loss: 0.0888 - val_accuracy: 0.
Epoch 13/20
30/30 [==============================] - 1s 39ms/step - loss: 0.0223 - accuracy: 0.9801 - val_loss: 0.0908 - val_accuracy: 0.
Epoch 14/20
30/30 [==============================] - 1s 40ms/step - loss: 0.0196 - accuracy: 0.9832 - val_loss: 0.0953 - val_accuracy: 0.
Epoch 15/20
30/30 [==============================] - 2s 59ms/step - loss: 0.0181 - accuracy: 0.9855 - val_loss: 0.0925 - val_accuracy: 0.
Epoch 16/20
30/30 [==============================] - 2s 56ms/step - loss: 0.0161 - accuracy: 0.9878 - val_loss: 0.0938 - val_accuracy: 0.
Epoch 17/20
30/30 [==============================] - 1s 37ms/step - loss: 0.0149 - accuracy: 0.9877 - val_loss: 0.0969 - val_accuracy: 0.
Epoch 18/20
30/30 [==============================] - 1s 39ms/step - loss: 0.0132 - accuracy: 0.9904 - val_loss: 0.0994 - val_accuracy: 0.
Epoch 19/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0127 - accuracy: 0.9899 - val_loss: 0.0976 - val_accuracy: 0.
Epoch 20/20
30/30 [==============================] - 1s 37ms/step - loss: 0.0113 - accuracy: 0.9905 - val_loss: 0.1001 - val_accuracy: 0.
```

Plotting training and testing loss

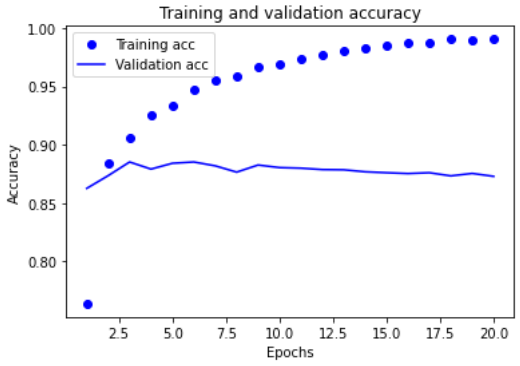
```
#Losses
historyp3 = history3.history
loss_values = historyp3["loss"]
val_loss_values = historyp3["val_loss"]
```

```
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", label="Training loss")
plt.plot(epochs, val_loss_values, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



Here, the minimum validation loss is observed in 3rd epoch.

```
#Accuracy
plt.clf()
acc = historyp3["accuracy"]
val_acc = historyp3["val_accuracy"]
plt.plot(epochs, acc, "bo", label="Training acc")
plt.plot(epochs, val_acc, "b", label="Validation acc")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



Maximum accuracy is seen in 2nd and 3rd epochs

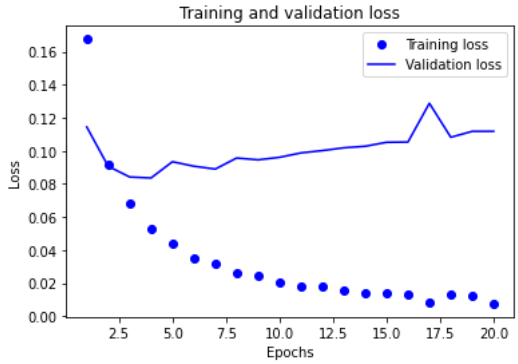
4. Using the tanh activation

```
#tanh activation function model
model_4 = keras.Sequential([
    layers.Dense(16, activation="tanh"),
    layers.Dense(16, activation="tanh"),
    layers.Dense(1, activation="sigmoid")
])

#RMSProp and binary cross entropy
model_4.compile(optimizer="rmsprop",
              loss="mse",
              metrics=["accuracy"])
```

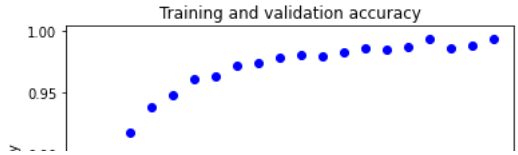
```
# Fitting model with 20 epochs and 512 batch size
history4 = model_4.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/20
30/30 [==============================] - 3s 77ms/step - loss: 0.1676 - accuracy: 0.7797 - val_loss: 0.1144 - val_accuracy: 0.
Epoch 2/20
30/30 [==============================] - 1s 41ms/step - loss: 0.0920 - accuracy: 0.8945 - val_loss: 0.0907 - val_accuracy: 0.
Epoch 3/20
30/30 [==============================] - 1s 40ms/step - loss: 0.0681 - accuracy: 0.9179 - val_loss: 0.0842 - val_accuracy: 0.
Epoch 4/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0527 - accuracy: 0.9379 - val_loss: 0.0836 - val_accuracy: 0.
Epoch 5/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0443 - accuracy: 0.9479 - val_loss: 0.0934 - val_accuracy: 0.
Epoch 6/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0353 - accuracy: 0.9611 - val_loss: 0.0907 - val_accuracy: 0.
Epoch 7/20
30/30 [==============================] - 1s 36ms/step - loss: 0.0318 - accuracy: 0.9633 - val_loss: 0.0890 - val_accuracy: 0.
Epoch 8/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0259 - accuracy: 0.9717 - val_loss: 0.0956 - val_accuracy: 0.
Epoch 9/20
30/30 [==============================] - 1s 39ms/step - loss: 0.0246 - accuracy: 0.9734 - val_loss: 0.0945 - val_accuracy: 0.
Epoch 10/20
30/30 [==============================] - 1s 45ms/step - loss: 0.0206 - accuracy: 0.9783 - val_loss: 0.0961 - val_accuracy: 0.
Epoch 11/20
30/30 [==============================] - 2s 54ms/step - loss: 0.0180 - accuracy: 0.9807 - val_loss: 0.0987 - val_accuracy: 0.
Epoch 12/20
30/30 [==============================] - 1s 47ms/step - loss: 0.0182 - accuracy: 0.9794 - val_loss: 0.1001 - val_accuracy: 0.
Epoch 13/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0156 - accuracy: 0.9829 - val_loss: 0.1018 - val_accuracy: 0.
Epoch 14/20
30/30 [==============================] - 1s 39ms/step - loss: 0.0142 - accuracy: 0.9853 - val_loss: 0.1028 - val_accuracy: 0.
Epoch 15/20
30/30 [==============================] - 1s 39ms/step - loss: 0.0137 - accuracy: 0.9849 - val_loss: 0.1052 - val_accuracy: 0.
Epoch 16/20
30/30 [==============================] - 1s 39ms/step - loss: 0.0130 - accuracy: 0.9867 - val_loss: 0.1053 - val_accuracy: 0.
Epoch 17/20
30/30 [==============================] - 1s 40ms/step - loss: 0.0081 - accuracy: 0.9928 - val_loss: 0.1288 - val_accuracy: 0.
Epoch 18/20
30/30 [==============================] - 1s 39ms/step - loss: 0.0134 - accuracy: 0.9853 - val_loss: 0.1082 - val_accuracy: 0.
Epoch 19/20
30/30 [==============================] - 1s 37ms/step - loss: 0.0121 - accuracy: 0.9874 - val_loss: 0.1118 - val_accuracy: 0.
Epoch 20/20
30/30 [==============================] - 1s 37ms/step - loss: 0.0075 - accuracy: 0.9931 - val_loss: 0.1118 - val_accuracy: 0.
```

```
historyp4 = history4.history
loss_values = historyp4["loss"]
val_loss_values = historyp4["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", label="Training loss")
plt.plot(epochs, val_loss_values, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()

plt.clf()
acc = historyp4["accuracy"]
val_acc = historyp4["val_accuracy"]
plt.plot(epochs, acc, "bo", label="Training acc")
plt.plot(epochs, val_acc, "b", label="Validation acc")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
```




```
<matplotlib.legend.Legend at 0x7f4805942fa0>
```



Maximum validation accuracy is seen in 3rd epoch where as Minimum validation loss is also seen in 3rd epoch

```
results
```

```
[0.28187671303749084, 0.8885200023651123]
```

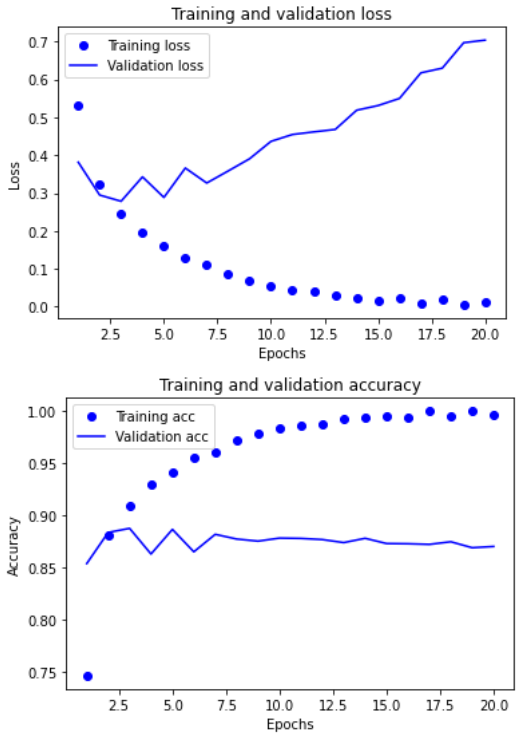
5. Using technique we studied in class

```
model_5 = keras.Sequential([
    layers.Dense(32, activation="relu"),
    layers.Dropout(0.2),
    layers.Dense(64, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])


#RMSProp and binary cross entropy
model_5.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])


# Fitting model with 20 epochs and 512 batch size
history5 = model_5.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/20
30/30 [==============================] - 3s 68ms/step - loss: 0.5308 - accuracy: 0.7466 - val_loss: 0.3815 - val_accuracy: 0.
Epoch 2/20
30/30 [==============================] - 1s 49ms/step - loss: 0.3213 - accuracy: 0.8803 - val_loss: 0.2947 - val_accuracy: 0.
Epoch 3/20
30/30 [==============================] - 2s 71ms/step - loss: 0.2454 - accuracy: 0.9091 - val_loss: 0.2783 - val_accuracy: 0.
Epoch 4/20
30/30 [==============================] - 2s 58ms/step - loss: 0.1975 - accuracy: 0.9291 - val_loss: 0.3427 - val_accuracy: 0.
Epoch 5/20
30/30 [==============================] - 1s 45ms/step - loss: 0.1598 - accuracy: 0.9411 - val_loss: 0.2884 - val_accuracy: 0.
Epoch 6/20
30/30 [==============================] - 1s 45ms/step - loss: 0.1287 - accuracy: 0.9553 - val_loss: 0.3660 - val_accuracy: 0.
Epoch 7/20
30/30 [==============================] - 1s 46ms/step - loss: 0.1122 - accuracy: 0.9598 - val_loss: 0.3265 - val_accuracy: 0.
Epoch 8/20
30/30 [==============================] - 1s 47ms/step - loss: 0.0854 - accuracy: 0.9714 - val_loss: 0.3582 - val_accuracy: 0.
Epoch 9/20
30/30 [==============================] - 1s 47ms/step - loss: 0.0693 - accuracy: 0.9777 - val_loss: 0.3905 - val_accuracy: 0.
Epoch 10/20
30/30 [==============================] - 1s 45ms/step - loss: 0.0556 - accuracy: 0.9824 - val_loss: 0.4364 - val_accuracy: 0.
Epoch 11/20
30/30 [==============================] - 2s 53ms/step - loss: 0.0455 - accuracy: 0.9850 - val_loss: 0.4545 - val_accuracy: 0.
Epoch 12/20
30/30 [==============================] - 2s 80ms/step - loss: 0.0405 - accuracy: 0.9873 - val_loss: 0.4615 - val_accuracy: 0.
Epoch 13/20
```

```
30/30 [==============================] - 1s 45ms/step - loss: 0.0291 - accuracy: 0.9915 - val_loss: 0.4679 - val_accuracy: 0.
Epoch 14/20
30/30 [==============================] - 1s 46ms/step - loss: 0.0243 - accuracy: 0.9935 - val_loss: 0.5186 - val_accuracy: 0.
Epoch 15/20
30/30 [==============================] - 1s 46ms/step - loss: 0.0171 - accuracy: 0.9951 - val_loss: 0.5310 - val_accuracy: 0.
Epoch 16/20
30/30 [==============================] - 1s 45ms/step - loss: 0.0211 - accuracy: 0.9934 - val_loss: 0.5498 - val_accuracy: 0.
Epoch 17/20
30/30 [==============================] - 1s 47ms/step - loss: 0.0078 - accuracy: 0.9990 - val_loss: 0.6173 - val_accuracy: 0.
Epoch 18/20
30/30 [==============================] - 1s 46ms/step - loss: 0.0177 - accuracy: 0.9945 - val_loss: 0.6297 - val_accuracy: 0.
Epoch 19/20
30/30 [==============================] - 1s 47ms/step - loss: 0.0054 - accuracy: 0.9993 - val_loss: 0.6969 - val_accuracy: 0.
Epoch 20/20
30/30 [==============================] - 2s 71ms/step - loss: 0.0134 - accuracy: 0.9961 - val_loss: 0.7037 - val_accuracy: 0.
```

```
#Losses
historyp5 = history5.history
loss_values = historyp5["loss"]
val_loss_values = historyp5["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", label="Training loss")
plt.plot(epochs, val_loss_values, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()

#Accuracy
plt.clf()
acc = historyp5["accuracy"]
val_acc = historyp5["val_accuracy"]
plt.plot(epochs, acc, "bo", label="Training acc")
plt.plot(epochs, val_acc, "b", label="Validation acc")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



Maximum accuracy and minimum validation loss is seen in 3rd epoch

```
results
```

```
[0.28187671303749084, 0.8885200023651123]
```