Cosmic Classifier:

A Deep Learning Approach to Galaxy Classification

1. Introduction: Defining the problem and assembling a dataset

In the vast expanse of our cosmic tapestry, galaxies stand as celestial entities with diverse forms and structures. Understanding and classifying these galaxies based on their shapes—whether they exhibit the graceful spirals or the more enigmatic ellipticals—is a crucial task in the realm of astronomy. This project, aptly named **"Cosmic Classifier,"** embarks on the mission to decipher the cosmic morphology, offering a systematic approach to the classification of galaxies. The challenge lies in the sheer volume and complexity of astronomical data. As telescopes capture images of galaxies scattered across the universe, the need for automated classification methods becomes increasingly apparent. Identifying the structural characteristics of galaxies is not only a key aspect of astronomical research but also a fundamental step towards unraveling the mysteries of the universe's evolution.

The primary goal of the "Cosmic Classifier" project is to develop a robust and efficient system for the classification of galaxies using galaxy images into two primary categories: spiral and elliptical. By harnessing the power of deep learning and image analysis, this project seeks to create a tool that can navigate the intricacies of galactic forms, providing astronomers and researchers with a powerful means to streamline their analyses. Through this project, we aim to contribute to the broader understanding of our cosmic surroundings and foster advancements in the field of astronomy.

1.1 Domain-specific Area

The quest to classify galaxies has undergone a transformative shift with the advent of machine learning, particularly neural networks. Traditionally reliant on manual classification methods, such as citizen science initiatives exemplified by Galaxy Zoo (Lintott et al., 2008), recent advancements have propelled the field toward automated solutions. One noteworthy study by Dieleman et al. (2015) delved into the application of convolutional neural networks (CNNs) for morphological classification. Their work showcased the ability of deep learning models to extract intricate features directly from pixel data, rivaling the accuracy of traditional methods.

Transfer learning, a technique demonstrating the transfer of knowledge from pre-trained models, has found application in galaxy classification. Huertas-Company et al. (2015) explored the use of pre-trained CNNs on ImageNet for fine-tuning on specific astronomical tasks. This approach demonstrated the efficacy of leveraging previously acquired knowledge to enhance the accuracy of galaxy classification models, highlighting the potential for knowledge transfer in image analysis within the astronomical domain.

In a broader context, the intersection of citizen science and machine learning has been investigated by Barchi et al. (2020). Their study integrated deep learning models into the Galaxy Zoo project, fostering collaboration between citizen scientists and machine learning algorithms. This fusion of human expertise and automated classification mechanisms not only improved accuracy but also allowed for scalable and efficient analyses of vast datasets.

Despite the strides made in the field, Khan et al. (2019) provided a comprehensive review outlining the challenges and opportunities associated with applying neural networks to large-scale galaxy surveys. Addressing issues such as imbalanced datasets, interpretability of deep models, and the necessity for robust architectures, the review underscored the need for ongoing research and collaboration between astronomers and machine learning experts. In essence, while neural networks have substantially advanced galaxy classification, challenges persist, and the field remains dynamic with continuous refinements and collaborations on the horizon.

1.2 Objectives

The objectives of the "Cosmic Classifier" project are designed to promote advancements in automated galaxy shape classification. The primary goals include developing a robust machine learning model that excels in discerning intricate patterns and structures inherent in astronomical images. The project emphasizes the optimization of the model for generalization across diverse datasets, ensuring scalability and efficiency in handling vast volumes of astronomical data generated by modern observatories.

User-centric objectives focus on creating an accessible and user-friendly interface for astronomers, promoting seamless integration into their workflow. Additionally, the project aims to enhance interpretability, providing insights into the decision-making process of the model. Encouraging collaboration and open science, the project is set to be open-source, inviting researchers to contribute, share insights, and collectively advance our understanding of galactic structures.

Continuous improvement is embedded in the project's objectives through a feedback loop, enabling astronomers to contribute input on misclassifications and uncertainties. Comparative analyses with existing classifications and alternative methods will validate the reliability and effectiveness of the "Cosmic Classifier." Ultimately, the project aspires to contribute to astronomical knowledge, providing a reliable tool for automated galaxy shape classification and unlocking new possibilities for research and exploration of the application of machine learning techniques in the field of astrophysics.

1.3 Dataset

1.3. Dataset

The dataset utilized in the "Cosmic Classifier" project originates from the **Galaxy Zoo project**, a groundbreaking initiative that provides visual morphological classifications for nearly one million galaxies extracted from the Sloan Digital Sky Survey (SDSS). This extensive dataset is the result of collaborative efforts involving more than **100,000 volunteers** who participated in the Galaxy Zoo project by visually inspecting and classifying galaxies through an online platform. This inclusive approach, involving the general public, has led to over 4 × 10^7 individual classifications, contributing to the creation of a robust morphological catalogue.

Motivated by the need to obtain accurate morphological classifications without the introduction of biases associated with proxies such as color, concentration, or structural parameters, the Galaxy Zoo project has demonstrated its effectiveness in generating reliable morphological data. The classifications performed by the volunteers have been found to be consistent with those made by professional astronomers on subsets of SDSS galaxies, affirming the reliability of the dataset. The dataset chosen from this project is the **Galaxy Zoo2 dataset**. This dataset consists of a total of **243,434** images, each capturing the unique morphology of galaxies observed in the Sloan Digital Sky Survey. These images serve as the foundation for the "Cosmic Classifier" project, aiming to leverage advanced machine learning and image analysis techniques for the automated classification of galaxies into two primary categories: spiral and elliptical. Along with these images are two datasets containing the image information, features and galaxy labels of this data.

The extensive participation of volunteers in the Galaxy Zoo project not only facilitated the creation of this valuable dataset but also underscores the collaborative nature of scientific endeavors in the exploration of our cosmic surroundings. The individual contributions of more than 100,000 volunteers who participated in the Galaxy Zoo project are duly acknowledged, highlighting the collective effort that has made this dataset a cornerstone for advancing our understanding of galactic structures.

2. Choosing a measure of success

A good measure of success for the "Cosmic Classifier" would be **accuracy**, particularly given the binary nature of the classification task (spiral or elliptical galaxies) and the balanced dataset. Accuracy is a straightforward metric that measures the proportion of correctly classified instances out of the total instances. Accuracy serves as a pivotal measure of success for the "Cosmic Classifier" project due to its intuitive interpretation, alignment with project goals, balanced evaluation, and user-friendly nature. With a clear indication of the percentage of correctly classified galaxies, accuracy simplifies the assessment of the model's performance, ensuring easy communication and understanding.

The choice of accuracy aligns well with the higher-level goal of the project, which is to provide an efficient and reliable tool for automated galaxy shape classification. he project's primary objective, the accurate classification of galaxies into spiral and elliptical categories, is directly addressed by maximizing accuracy. Additionally, the balanced evaluation provided by accuracy, considering both true positives and true negatives, proves crucial in the context of a binary classification task where both classes hold equal significance. This user-friendly performance metric contributes to the project's transparency and accessibility for researchers and stakeholders alike, facilitating a comprehensive evaluation of the "Cosmic Classifier's" success in achieving its primary goals.

3. Deciding on an evaluation protocol

Among the commonly used protocols—maintaining a hold-out validation set, K-fold cross-validation, and iterated K-fold validation—the most fitting choice depends on various factors such as the size of the dataset, available computational resources, and the need for robust performance estimation. Given the importance of obtaining a robust evaluation in the "Cosmic Classifier" project and considering the potential variability in the dataset, K-fold cross-validation or iterated K-fold validation would be preferable. These protocols offer more reliable estimates of the model's generalization performance, especially when dealing with a moderate dataset size. The choice between the two depends on computational resources, with K-fold cross-validation being more computationally efficient, while iterated K-fold validation provides additional stability in performance estimation.

However, due to restraints on computational resources, **maintaining a hold-out validation** is used for this project. This works by splitting the dataset into two subsets: one for training the model and another for evaluating its performance, allowing the model to learn patterns from one portion of the data and be evaluated on an independent subset. This method is good for its simplicity, efficiency, and ability to simulate real-world scenario. It is also suitable for this dataset that is moderately sized. While it has its limitations, such as sensitivity to specific data splits, hold-out validation remains a resource-efficient and practical choice, striking a balance between model assessment accuracy and computational requirements.

---

4. Preparing the data

The main goal for the preparation of this data was to label the images as elliptical or spiral given their image information from two csv files. The first dataset contains many features with particular interest in three features relevant for identifying the images and their labels- **objid**, **sample**, **gz2_class**. The objid column contains the Data Release 7 (DR7) object ID for each galaxy, the sampling contains a string indicating the subsampling of the galaxy and the gz2_class column contains the class of the galaxy (spiral, eliiptical and unknown) and its subclass. The second dataset contains the objid, sample columns and asset_id column that contains an integer that corresponds to the filename of the image in the zipped file. The datasets are merged based and three features, **objid**, **asset_id**, **gz2_class** of interest are extracted. The g2z_class column was changed to just the class of the galaxy instead of the class and subclass, since the subclass is not relevant to this classification. The "unknown" class from the gz2_class column was also dropped as this is intended to be a binary classification and the number of the data points that belong to the unknown class is few and would create a class imbalance. After this, the the asset_id column is and g2z_class column are used to map images to their labels, images without correspondiing objid value are skipped. A subset of **70,000** images and their labels were chosen. The resulting images and labels are saved as arrays using numpy so they can be loaded and used without having to re-do data preprocessing. A glimpse into the images and their labels is shown below.

In [2]:
```
import os
import cv2
import seaborn
import numpy as np
import pandas as pd
from keras import models
from keras import layers
from keras import losses
from keras import metrics
from keras import optimizers
from keras import regularizers
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import train_test_split
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.layers import Dense, Flatten, Dropout, Conv2D, MaxPooling2D
```

```
Using TensorFlow backend.
```

In [2]:
```
# Import first dataset
file_path = "/Users/fisayo/Downloads/NN_CW"
data = pd.read_csv(file_path + '/gz2_hart16.csv')
data.head()
```

Out[2]:

	dr7objid	ra	dec	rastring	decstring	sample	gz2_class	total_classifications	total_votes	t01_smooth_or_fe
0	587732591714893851	179.042984	60.522518	11:56:10.32	+60:31:21.1	original	Sc+t	45	342	
1	588009368545984617	135.084396	52.494240	09:00:20.26	+52:29:39.3	original	Sb+t	42	332	
2	587732484359913515	183.371979	50.741508	12:13:29.27	+50:44:29.4	original	Ei	36	125	
3	587741723357282317	186.251953	28.558598	12:25:00.47	+28:33:31.0	original	Sc+t	28	218	
4	587738410866966577	161.086395	14.084465	10:44:20.73	+14:05:04.1	original	Er	43	151	

5 rows × 231 columns

In [3]:
```
# Import second dataset
data2 = pd.read_csv(file_path + '/gz2_filename_mapping.csv')
data2.head()
```

Out[3]:

	objid	sample	asset_id
0	587722981736120347	original	1
1	587722981736579107	original	2
2	587722981741363294	original	3
3	587722981741363323	original	4
4	587722981741559888	original	5

In [4]:
```
# Change id column name to match the column name in the other datafrma
data.rename(columns={'dr7objid' : 'objid'}, inplace = True)
data.head()
```

Out[4]:

	objid	ra	dec	rastring	decstring	sample	gz2_class	total_classifications	total_votes	t01_smooth_or_fe
0	587732591714893851	179.042984	60.522518	11:56:10.32	+60:31:21.1	original	Sc+t	45	342	
1	588009368545984617	135.084396	52.494240	09:00:20.26	+52:29:39.3	original	Sb+t	42	332	
2	587732484359913515	183.371979	50.741508	12:13:29.27	+50:44:29.4	original	Ei	36	125	
3	587741723357282317	186.251953	28.558598	12:25:00.47	+28:33:31.0	original	Sc+t	28	218	
4	587738410866966577	161.086395	14.084465	10:44:20.73	+14:05:04.1	original	Er	43	151	

5 rows × 231 columns

In [5]:
```
# merge datasets on the common column (only common rows will be included)
merged_df = pd.merge(data, data2, on='objid')

# extract columns A, B, and C from the merged dataset
result_df = merged_df[['objid', 'asset_id', 'gz2_class']]

# save as csv to local machine to be used later
```

```
result_df.to_csv(file_path + '/result_df.csv', index=False)

result_df.head()
```

Out[5]:

	objid	asset_id	gz2_class
0	587732591714893851	58957	Sc+t
1	588009368545984617	193641	Sb+t
2	587732484359913515	55934	Ei
3	587741723357282317	158501	Sc+t
4	587738410866966577	110939	Er

In [7]:
```
# Creat a copy of the dataframe
result2 = result_df.copy()
```

In [10]:
```
# Check the number of each galaxy class
print('Number of spiral galaxies ', len(result2.loc[result2['gz2_class'].str.startswith('S')]))
print('Number of elliptical galaxies ', len(result2.loc[result2['gz2_class'].str.startswith('E')]))
```

```
Number of spiral galaxies  141430
Number of elliptical galaxies  97670
```

In [11]:
```
# Remove unknown or other class that is not spiral or elliptical
other = [un for un in result2['gz2_class'].unique() if not un.startswith(('S', 'E'))]
print(other)
print(len(result2[result2['gz2_class'] == 'A']))
```

```
['A']
595
```

In [12]:
```
# Change galaxy class and subclass categories to just galaxy class
result2.loc[result2['gz2_class'].str.lower().str.startswith('s'), 'gz2_class'] = 'spiral'
result2.loc[result2['gz2_class'].str.lower().str.startswith('e'), 'gz2_class'] = 'elliptical'
```

In [13]:
```
# Keep only the spiral and elliptical classifications
result2 = result2[(result2['gz2_class'] == 'spiral') | (result2['gz2_class'] == 'elliptical')]
result2.head()
```

Out[13]:

	objid	asset_id	gz2_class
0	587732591714893851	58957	spiral
1	588009368545984617	193641	spiral
2	587732484359913515	55934	elliptical
3	587741723357282317	158501	spiral
4	587738410866966577	110939	elliptical

In [14]:
```
# Check length of dataframe
len(result2)
```

Out[14]:
```
239100
```

In [16]:
```
# taking a sample of 70000 data points to test the model with
sample_gz = result2.head(70000)

# dropping the 'objid' column since it is no longer needed
sample_gz = sample_gz.drop(columns=['objid'])

# # adding .jpg to the rows in the 'asset_id' column so the images are properly matched
sample_gz['asset_id'] = sample_gz['asset_id'].astype(str) + '.jpg'

# replacing the spiral and elliptical strings to 0 and 1
sample_gz['gz2_class'] = sample_gz['gz2_class'].replace({'elliptical': 0, 'spiral': 1})

# # saving the dataset
sample_gz.to_csv(file_path + '/sample_gz.csv', index=False)

sample_gz.head()
```

Out[16]:

	asset_id	gz2_class
0	58957.jpg	1
1	193641.jpg	1
2	55934.jpg	0
3	158501.jpg	1
4	110939.jpg	0

In [18]:
```
# Check the length of each class
print('Number of spiral galaxies ', len(sample_gz.loc[sample_gz['gz2_class']==1]))
```

```python
print('Number of elliptical galaxies ', len(sample_gz.loc[sample_gz['gz2_class']==0]))
```

```
Number of spiral galaxies  40959
Number of elliptical galaxies  29041
```

In [19]:
```python
# Checking for null values
sample_gz.isnull().sum()
```

Out[19]:
```
asset_id     0
gz2_class    0
dtype: int64
```

In [43]:
```python
images = []
labels = []

# Replace 'path/to/your/dataset/images' with the actual path to your image folder
image_folder = file_path + '/images'

for index, row in sample_gz.iterrows():
    filename = row['asset_id']
    label = row['gz2_class']

    # Construct the path to the image
    image_path = os.path.join(image_folder, filename)

    try:
        # Read and resize the image
        image = cv2.imread(image_path)
        if image is None:
            print(f"Skipping {filename} - Unable to read the image.")
            continue

        image = cv2.resize(image, (140, 140))  # Adjust the size as needed

        images.append(image)
        labels.append(label)
    except Exception as e:
        print(f"Error processing {filename}: {e}")

images = np.array(images)
labels = np.array(labels)

# Some images are not in the other datasets and some are png and other formats, we skip those
```

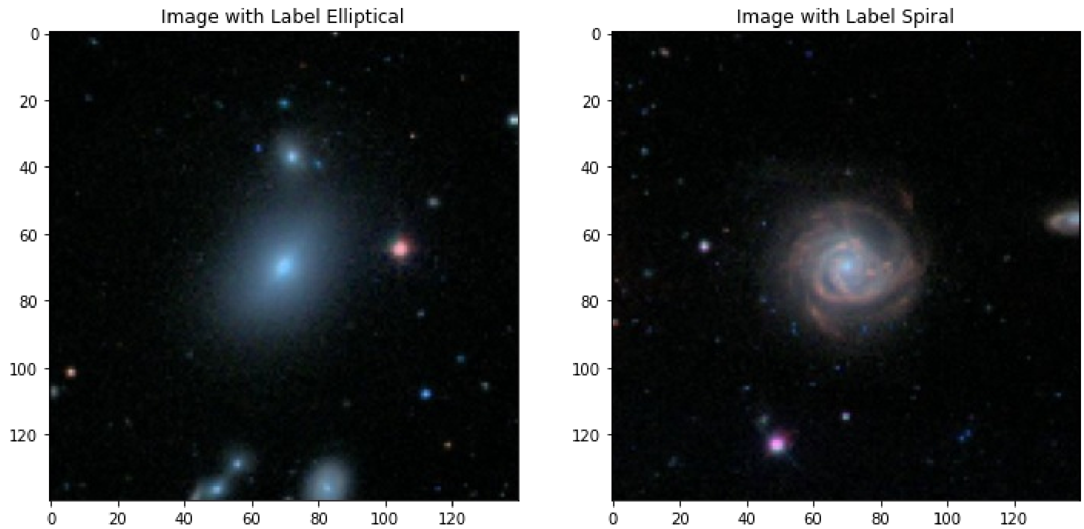
```
Skipping 26603.jpg - Unable to read the image.
[ WARN:0@2681.667] global /private/var/folders/sy/f16zz6x50xz3113nwtb9bvq00000gp/T/abs_5a1v4y7k9y/croot/opencv-suite_1676472757237/work/modules/imgcodecs/src/loadsave.cpp (239) findDecoder imread_('/Users/fisayo/Downloads/NN_CW/images/26603.jpg'): can't open/read file: check file path/integrity
Skipping 215414.jpg - Unable to read the image.
[ WARN:0@2717.462] global /private/var/folders/sy/f16zz6x50xz3113nwtb9bvq00000gp/T/abs_5a1v4y7k9y/croot/opencv-suite_1676472757237/work/modules/imgcodecs/src/loadsave.cpp (239) findDecoder imread_('/Users/fisayo/Downloads/NN_CW/images/215414.jpg'): can't open/read file: check file path/integrity
Skipping 59270.jpg - Unable to read the image.
[ WARN:0@2732.512] global /private/var/folders/sy/f16zz6x50xz3113nwtb9bvq00000gp/T/abs_5a1v4y7k9y/croot/opencv-suite_1676472757237/work/modules/imgcodecs/src/loadsave.cpp (239) findDecoder imread_('/Users/fisayo/Downloads/NN_CW/images/59270.jpg'): can't open/read file: check file path/integrity
Skipping 249103.jpg - Unable to read the image.
[ WARN:0@2741.075] global /private/var/folders/sy/f16zz6x50xz3113nwtb9bvq00000gp/T/abs_5a1v4y7k9y/croot/opencv-suite_1676472757237/work/modules/imgcodecs/src/loadsave.cpp (239) findDecoder imread_('/Users/fisayo/Downloads/NN_CW/images/249103.jpg'): can't open/read file: check file path/integrity
Skipping 99870.jpg - Unable to read the image.
[ WARN:0@2751.982] global /private/var/folders/sy/f16zz6x50xz3113nwtb9bvq00000gp/T/abs_5a1v4y7k9y/croot/opencv-suite_1676472757237/work/modules/imgcodecs/src/loadsave.cpp (239) findDecoder imread_('/Users/fisayo/Downloads/NN_CW/images/99870.jpg'): can't open/read file: check file path/integrity
Skipping 219381.jpg - Unable to read the image.
[ WARN:0@2760.106] global /private/var/folders/sy/f16zz6x50xz3113nwtb9bvq00000gp/T/abs_5a1v4y7k9y/croot/opencv-suite_1676472757237/work/modules/imgcodecs/src/loadsave.cpp (239) findDecoder imread_('/Users/fisayo/Downloads/NN_CW/images/219381.jpg'): can't open/read file: check file path/integrity
Skipping 50643.jpg - Unable to read the image.
[ WARN:0@2777.414] global /private/var/folders/sy/f16zz6x50xz3113nwtb9bvq00000gp/T/abs_5a1v4y7k9y/croot/opencv-suite_1676472757237/work/modules/imgcodecs/src/loadsave.cpp (239) findDecoder imread_('/Users/fisayo/Downloads/NN_CW/images/50643.jpg'): can't open/read file: check file path/integrity
```

In [44]:
```python
# Save images and labels as numpy arrays
np.save('images.npy', images)
np.save('labels.npy', labels)
```

In [4]:
```python
# Load images and labels from binary files
images = np.load('images.npy')
labels = np.load('labels.npy')
```

In [3]:
```python
# Set the target labels
target_label_1 = 0
target_label_2 = 1

# Find the index of the first image with the first target label
```

```
index_1 = np.argmax(labels == target_label_1)

# Find the index of the first image with the second target label
index_2 = np.argmax(labels == target_label_2)

# Plot the images side by side
plt.figure(figsize=(12, 6))

# Plot image with the first target label
plt.subplot(1, 2, 1)
plt.imshow(images[index_1])
plt.title(f'Image with Label Elliptical')

# Plot image with the second target label
plt.subplot(1, 2, 2)
plt.imshow(images[index_2])
plt.title(f'Image with Label Spiral')

plt.show()
```



5. Developing a model that does better than a baseline

To develop a model better than a baseline model, a simple model is built based on a template from Chollet, 2019. The network is a simple stack of fully connected (Dense) layers with relu activations. Two intermediate layers with 16 hidden units each and a third layer that will output the scalar prediction regarding the classification of the galaxy. The intermediate layers will use relu as their activation function, and the final layer will use a sigmoid activation so as to output a probability (a score between 0 and 1, indicating how likely the sample is to have the target “1”: how likely the galaxy is to be spiral). RMSprop is used as the optimizer, and binary_crossentropy is used as the loss; these are the preferred configurations for binary classification. Accuracy is used as the metric because it provides a straightforward measure of the model's overall performance on the task of classifying galaxies into spiral or elliptical categories. The training and validation losses and accuracy are plotted to get a visual understanding of the models performance.

The model achieved an accuracy of 77% which is better than a baseline model for binary classification which is expected to have an accuracy of about 50%. The rising and falling validation loss and accuracy indicate that the learning rate is too high, which be causing the model to oscillate around the optimal point, causing fluctuations in the loss and accuracy.

In [5]:
```
# Normalise the images data to the range [0,1]
images = images.astype('float32') / 255.0

images2 = images.reshape((69993, 140 * 140 * 3))

# Splitting the data into training and testing sets
train_images, test_images, train_labels, test_labels = train_test_split(images2, labels, test_size=0.2, random_

print ('Training images shape: ', train_images.shape)
print ('Test images shape: ', test_images.shape)
```

In [22]:
```
# Create a validation set frorm the training set
val_images = train_images[:10000]
partial_train_images = train_images[10000:]
val_labels = train_labels[:10000]
partial_train_labels = train_labels[10000:]
```

In [6]:
```
def create_model(units, learning_rate=0.001, add_layers=0, regularization=None, dropout=None):
    """
    Create a neural network model with customizable architecture.

    Parameters:
```

```
    - units (int): Number of units/neurons in the hidden layers.
    - learning_rate (float, optional): Learning rate for the optimizer (default is 0.001).
    - add_layers (int, optional): Number of additional hidden layers to be added (default is 0).
    - regularization (str or None, optional): Regularization technique, e.g., 'l1' or 'l2', or None if no regul
    - dropout (float or None, optional): Dropout rate, representing the fraction of input units to drop (defaul

    Returns:
    - keras.models.Sequential: Compiled neural network model.

    """
    # model definition
    model = models.Sequential()

    if regularization is not None:
        reg = getattr(regularizers, regularization)(0.01)
    else:
        reg = None

    model.add(layers.Dense(units, activation='relu', input_shape=(140 * 140 * 3,)))
    model.add(layers.Dense(units, activation='relu', kernel_regularizer=reg))

    if dropout is not None:
        model.add(layers.Dropout(dropout))

    for _ in range(add_layers):
        model.add(layers.Dense(units, activation='relu', kernel_regularizer=reg))

    model.add(layers.Dense(1, activation='sigmoid'))

    model.compile(optimizer=optimizers.RMSprop(learning_rate=learning_rate),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

    return model
```

In [8]:

```
def history(model, epochs=20, batch_size=512):
    """
    Train a given model and return its training history.

    Parameters:
    - model (keras.models.Model): The Keras model to be trained.
    - epochs (int): The number of epochs for training the model. Default is 20.
    - batch_size (int): The batch size used during training. Default is 512.

    Returns:
    - dict: A dictionary containing the training history with keys 'loss', 'accuracy',
            'val_loss', and 'val_accuracy'. Each key corresponds to a list of values
            representing the respective metric's evolution during training.
    """

    history = model.fit(partial_train_images,
                        partial_train_labels,
                        epochs=epochs,
                        batch_size=batch_size,
                        validation_data=(val_images, val_labels))

    history_dict = history.history
    return history_dict
```

In [9]:

```
def plot_history(history_dict, loss_values, val_loss_values, acc_values, val_acc_values):
    """
    Plots training and validation loss, as well as training and validation accuracy, over epochs.

    Parameters:
    - history_dict (dict): The dictionary containing training history, typically obtained from model training.
    - loss_values (list): List of training loss values over epochs.
    - val_loss_values (list): List of validation loss values over epochs.
    - acc_values (list): List of training accuracy values over epochs.
    - val_acc_values (list): List of validation accuracy values over epochs.

    Returns:
    None

    The function generates a subplot with two plots: one for training and validation loss, and the other for
    training and validation accuracy. The x-axis represents the number of epochs, while the y-axes represent
    loss and accuracy values. The function utilizes matplotlib for plotting and displays the combined plot.
    """
    epochs = range(1, len(acc_values) + 1)

    plt.figure(figsize=(12, 6))

    # plotting the training and validation loss
    plt.subplot(1, 2, 1)  # 1 row, 2 columns, select the first plot
    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')
```

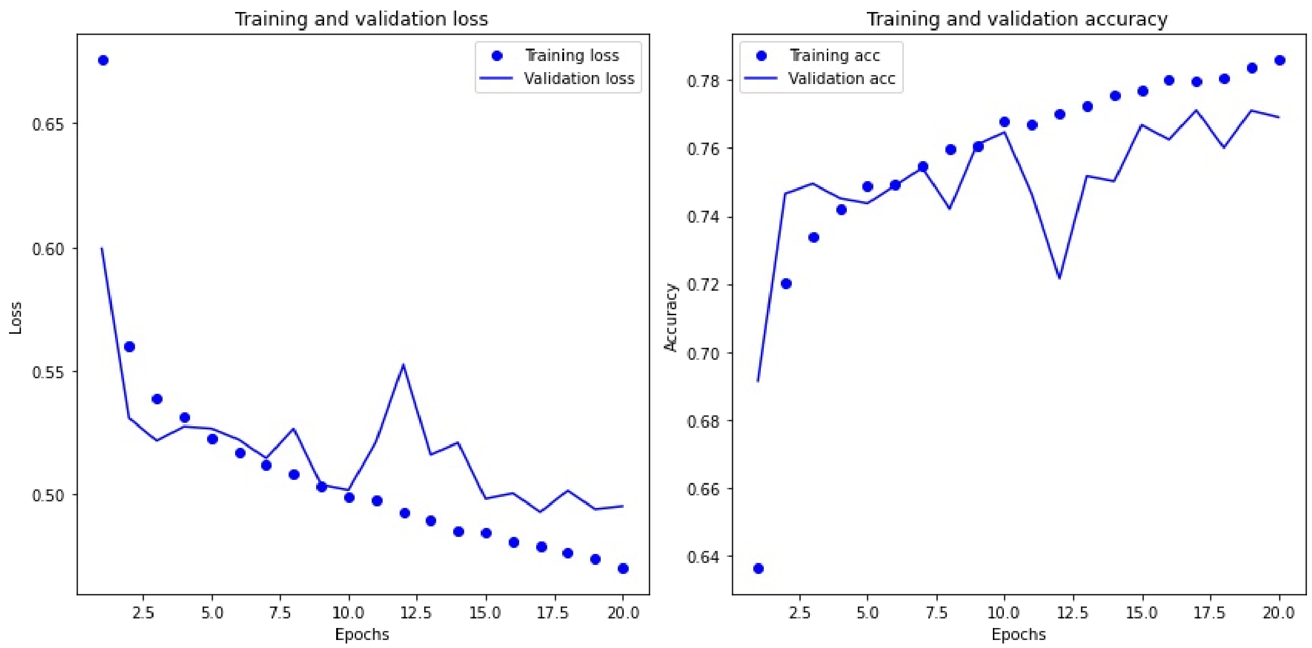
```python
    plt.legend()

    # plotting the training and validation accuracy
    plt.subplot(1, 2, 2)  # 1 row, 2 columns, select the second plot
    plt.plot(epochs, acc_values, 'bo', label='Training acc')
    plt.plot(epochs, val_acc_values, 'b', label='Validation acc')
    plt.title('Training and validation accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

    # Adjust layout to prevent clipping
    plt.tight_layout()

    # Show the combined plot
    plt.show()
```

In [9]:

```python
# Base Model
units = 16
learning_rate = 0.001

base_model = create_model(units, learning_rate)
base_history = history(base_model, 20, 512)

base_history_dict = base_history
base_loss_values = base_history_dict['loss']
base_val_loss_values = base_history_dict['val_loss']
base_acc_values = base_history_dict['accuracy']
base_val_acc_values = base_history_dict['val_accuracy']

plot_history(base_history_dict, base_loss_values, base_val_loss_values, base_acc_values, base_val_acc_values)
```

```
2024-01-14 13:30:30.479202: I tensorflow/core/platform/cpu_feature_guard.cc:145] This TensorFlow binary is opti
mized with Intel(R) MKL-DNN to use the following CPU instructions in performance critical operations:  SSE4.1 S
SE4.2
To enable them in non-MKL-DNN operations, rebuild TensorFlow with the appropriate compiler flags.
2024-01-14 13:30:30.483259: I tensorflow/core/common_runtime/process_util.cc:115] Creating new thread pool with
default inter op setting: 8. Tune using inter_op_parallelism_threads for best performance.
```

```
Train on 45994 samples, validate on 10000 samples
Epoch 1/20
45994/45994 [==============================] - 76s 2ms/step - loss: 0.6759 - accuracy: 0.6363 - val_loss: 0.599
5 - val_accuracy: 0.6914
Epoch 2/20
45994/45994 [==============================] - 74s 2ms/step - loss: 0.5603 - accuracy: 0.7203 - val_loss: 0.530
9 - val_accuracy: 0.7466
Epoch 3/20
45994/45994 [==============================] - 117s 3ms/step - loss: 0.5391 - accuracy: 0.7339 - val_loss: 0.52
18 - val_accuracy: 0.7495
Epoch 4/20
45994/45994 [==============================] - 83s 2ms/step - loss: 0.5313 - accuracy: 0.7419 - val_loss: 0.527
5 - val_accuracy: 0.7452
Epoch 5/20
45994/45994 [==============================] - 90s 2ms/step - loss: 0.5226 - accuracy: 0.7490 - val_loss: 0.526
7 - val_accuracy: 0.7437
Epoch 6/20
45994/45994 [==============================] - 78s 2ms/step - loss: 0.5174 - accuracy: 0.7494 - val_loss: 0.522
2 - val_accuracy: 0.7489
Epoch 7/20
45994/45994 [==============================] - 79s 2ms/step - loss: 0.5123 - accuracy: 0.7547 - val_loss: 0.514
7 - val_accuracy: 0.7542
Epoch 8/20
45994/45994 [==============================] - 75s 2ms/step - loss: 0.5083 - accuracy: 0.7598 - val_loss: 0.526
7 - val_accuracy: 0.7421
Epoch 9/20
45994/45994 [==============================] - 86s 2ms/step - loss: 0.5036 - accuracy: 0.7606 - val_loss: 0.504
1 - val_accuracy: 0.7610
Epoch 10/20
45994/45994 [==============================] - 78s 2ms/step - loss: 0.4988 - accuracy: 0.7679 - val_loss: 0.501
9 - val_accuracy: 0.7647
Epoch 11/20
45994/45994 [==============================] - 83s 2ms/step - loss: 0.4979 - accuracy: 0.7669 - val_loss: 0.521
1 - val_accuracy: 0.7464
Epoch 12/20
45994/45994 [==============================] - 74s 2ms/step - loss: 0.4927 - accuracy: 0.7701 - val_loss: 0.552
5 - val_accuracy: 0.7216
Epoch 13/20
45994/45994 [==============================] - 70s 2ms/step - loss: 0.4894 - accuracy: 0.7725 - val_loss: 0.516
1 - val_accuracy: 0.7518
Epoch 14/20
45994/45994 [==============================] - 66s 1ms/step - loss: 0.4854 - accuracy: 0.7756 - val_loss: 0.521
0 - val_accuracy: 0.7502
Epoch 15/20
45994/45994 [==============================] - 74s 2ms/step - loss: 0.4845 - accuracy: 0.7768 - val_loss: 0.498
3 - val_accuracy: 0.7668
Epoch 16/20
45994/45994 [==============================] - 66s 1ms/step - loss: 0.4811 - accuracy: 0.7799 - val_loss: 0.500
6 - val_accuracy: 0.7625
Epoch 17/20

45994/45994 [==============================] - 76s 2ms/step - loss: 0.4794 - accuracy: 0.7797 - val_loss: 0.493
0 - val_accuracy: 0.7712
Epoch 18/20
45994/45994 [==============================] - 83s 2ms/step - loss: 0.4766 - accuracy: 0.7808 - val_loss: 0.501
6 - val_accuracy: 0.7601
Epoch 19/20
45994/45994 [==============================] - 70s 2ms/step - loss: 0.4740 - accuracy: 0.7840 - val_loss: 0.494
1 - val_accuracy: 0.7711
Epoch 20/20
45994/45994 [==============================] - 87s 2ms/step - loss: 0.4703 - accuracy: 0.7861 - val_loss: 0.495
4 - val_accuracy: 0.7691
```



6. Scaling up: developing a model that overfits

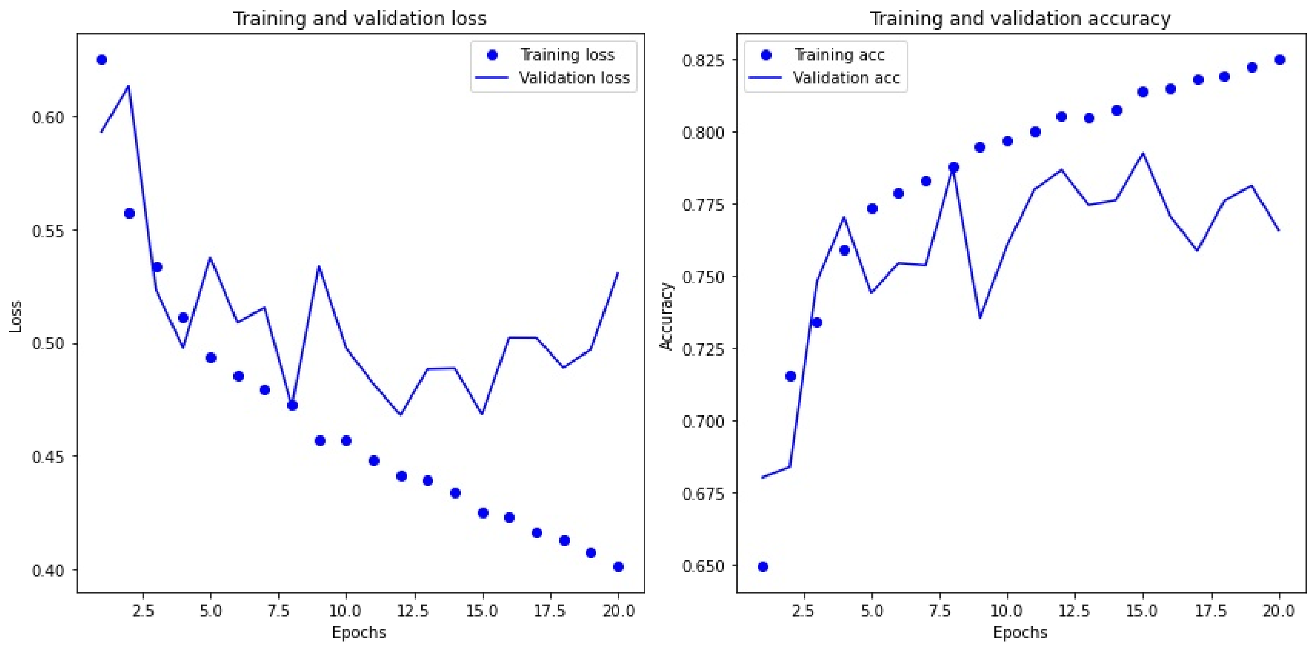
To develop a model prone to overfitting, experiments were conducted with various configurations of layers and units. Initially, an additional layer was introduced to the network, but it did not contribute to an improvement in accuracy. Subsequently, the number of units was increased from 16 to 32, resulting in a notable enhancement in network accuracy and a reduction in variation in both validation loss and accuracy. However, further increasing the units to 64 led to a decline in accuracy and an increase in variations in validation metrics. As a result, the decision was made to proceed with 32 units. Additional training epochs did not significantly enhance the model's accuracy. While the option to explore more epochs remains, it will be considered when there is stable validation accuracy and loss. Overall, a model prone to overfitting was successfully achieved.

6.1 Model 1: Adding layers

In [11]:

```
units = 16
learning_rate = 0.001
add_layers = 1

model_1 = create_model(units, learning_rate, add_layers)
m1_history = history(model_1, 20, 512)

m1_history_dict = m1_history
m1_loss_values = m1_history_dict['loss']
m1_val_loss_values = m1_history_dict['val_loss']
m1_acc_values = m1_history_dict['accuracy']
m1_val_acc_values = m1_history_dict['val_accuracy']

plot_history(m1_history_dict, m1_loss_values, m1_val_loss_values, m1_acc_values, m1_val_acc_values)
```

```
Train on 45994 samples, validate on 10000 samples
Epoch 1/20
45994/45994 [==============================] - 78s 2ms/step - loss: 0.6255 - accuracy: 0.6494 - val_loss: 0.593
3 - val_accuracy: 0.6802
Epoch 2/20
45994/45994 [==============================] - 79s 2ms/step - loss: 0.5573 - accuracy: 0.7153 - val_loss: 0.613
5 - val_accuracy: 0.6838
Epoch 3/20
45994/45994 [==============================] - 75s 2ms/step - loss: 0.5334 - accuracy: 0.7339 - val_loss: 0.523
4 - val_accuracy: 0.7482
Epoch 4/20
45994/45994 [==============================] - 68s 1ms/step - loss: 0.5113 - accuracy: 0.7590 - val_loss: 0.497
6 - val_accuracy: 0.7705
Epoch 5/20
45994/45994 [==============================] - 70s 2ms/step - loss: 0.4938 - accuracy: 0.7737 - val_loss: 0.537
7 - val_accuracy: 0.7441
Epoch 6/20
45994/45994 [==============================] - 75s 2ms/step - loss: 0.4857 - accuracy: 0.7791 - val_loss: 0.508
9 - val_accuracy: 0.7545
Epoch 7/20
45994/45994 [==============================] - 79s 2ms/step - loss: 0.4791 - accuracy: 0.7833 - val_loss: 0.515
4 - val_accuracy: 0.7537
Epoch 8/20
45994/45994 [==============================] - 87s 2ms/step - loss: 0.4729 - accuracy: 0.7882 - val_loss: 0.471
9 - val_accuracy: 0.7873
Epoch 9/20
45994/45994 [==============================] - 91s 2ms/step - loss: 0.4569 - accuracy: 0.7950 - val_loss: 0.533
8 - val_accuracy: 0.7355
Epoch 10/20
45994/45994 [==============================] - 84s 2ms/step - loss: 0.4571 - accuracy: 0.7970 - val_loss: 0.497
6 - val_accuracy: 0.7607
Epoch 11/20
45994/45994 [==============================] - 75s 2ms/step - loss: 0.4484 - accuracy: 0.8003 - val_loss: 0.481
8 - val_accuracy: 0.7799
Epoch 12/20
45994/45994 [==============================] - 76s 2ms/step - loss: 0.4415 - accuracy: 0.8053 - val_loss: 0.467
9 - val_accuracy: 0.7868
Epoch 13/20
45994/45994 [==============================] - 74s 2ms/step - loss: 0.4390 - accuracy: 0.8050 - val_loss: 0.488
3 - val_accuracy: 0.7746
Epoch 14/20
45994/45994 [==============================] - 75s 2ms/step - loss: 0.4337 - accuracy: 0.8077 - val_loss: 0.488
6 - val_accuracy: 0.7763
Epoch 15/20
45994/45994 [==============================] - 71s 2ms/step - loss: 0.4247 - accuracy: 0.8143 - val_loss: 0.468
3 - val_accuracy: 0.7926
Epoch 16/20
45994/45994 [==============================] - 65s 1ms/step - loss: 0.4228 - accuracy: 0.8153 - val_loss: 0.502
3 - val_accuracy: 0.7708
Epoch 17/20
45994/45994 [==============================] - 78s 2ms/step - loss: 0.4163 - accuracy: 0.8184 - val_loss: 0.502
2 - val_accuracy: 0.7587
Epoch 18/20
45994/45994 [==============================] - 84s 2ms/step - loss: 0.4125 - accuracy: 0.8196 - val_loss: 0.488
9 - val_accuracy: 0.7762
Epoch 19/20
45994/45994 [==============================] - 61s 1ms/step - loss: 0.4073 - accuracy: 0.8228 - val_loss: 0.496
9 - val_accuracy: 0.7813
Epoch 20/20
45994/45994 [==============================] - 65s 1ms/step - loss: 0.4010 - accuracy: 0.8253 - val_loss: 0.530
6 - val_accuracy: 0.7658
```

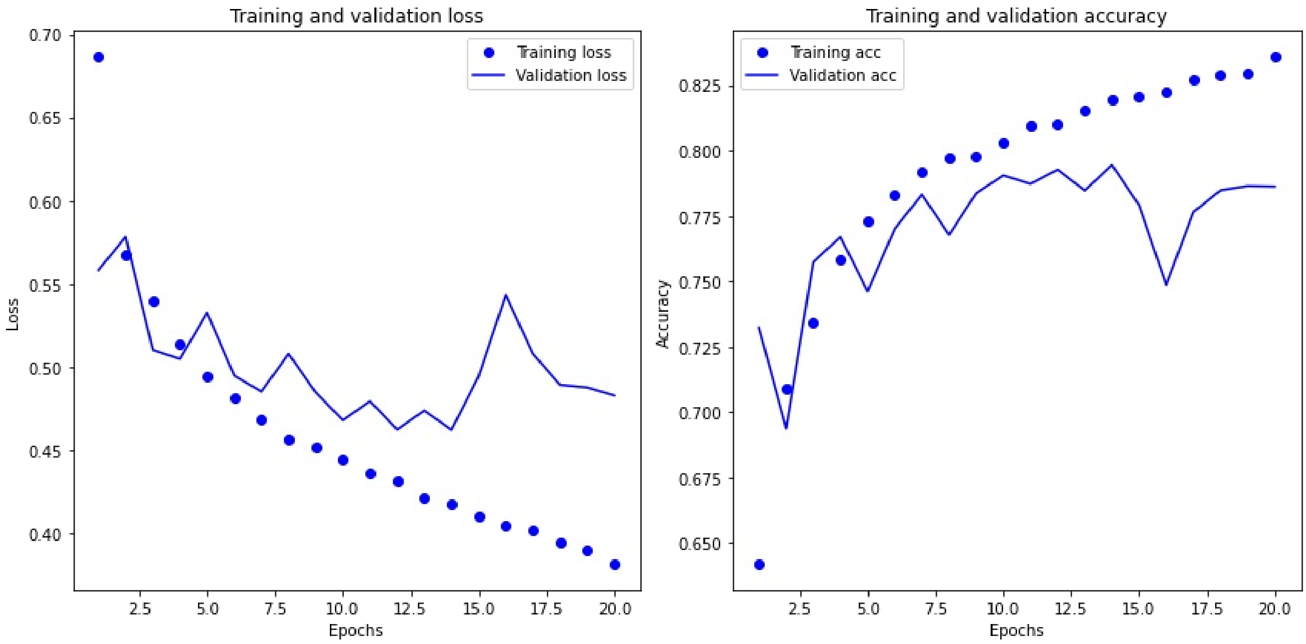


6.2 Model 2: Making the layers bigger (32 units)

In [12]:

```
units = 32
learning_rate = 0.001

model_2 = create_model(units, learning_rate, add_layers)
m2_history = history(model_2, 20, 512)

m2_history_dict = m2_history
m2_loss_values = m2_history_dict['loss']
m2_val_loss_values = m2_history_dict['val_loss']
m2_acc_values = m2_history_dict['accuracy']
m2_val_acc_values = m2_history_dict['val_accuracy']

plot_history(m2_history_dict, m2_loss_values, m2_val_loss_values, m2_acc_values, m2_val_acc_values)
```

```
Train on 45994 samples, validate on 10000 samples
Epoch 1/20
45994/45994 [==============================] - 97s 2ms/step - loss: 0.6868 - accuracy: 0.6418 - val_loss: 0.558
3 - val_accuracy: 0.7323
Epoch 2/20
45994/45994 [==============================] - 79s 2ms/step - loss: 0.5674 - accuracy: 0.7092 - val_loss: 0.578
3 - val_accuracy: 0.6938
Epoch 3/20
45994/45994 [==============================] - 78s 2ms/step - loss: 0.5394 - accuracy: 0.7342 - val_loss: 0.510
2 - val_accuracy: 0.7574
Epoch 4/20
45994/45994 [==============================] - 78s 2ms/step - loss: 0.5140 - accuracy: 0.7586 - val_loss: 0.504
8 - val_accuracy: 0.7670
Epoch 5/20
45994/45994 [==============================] - 62s 1ms/step - loss: 0.4941 - accuracy: 0.7728 - val_loss: 0.532
6 - val_accuracy: 0.7462
Epoch 6/20
45994/45994 [==============================] - 83s 2ms/step - loss: 0.4812 - accuracy: 0.7829 - val_loss: 0.494
8 - val_accuracy: 0.7701
Epoch 7/20
45994/45994 [==============================] - 84s 2ms/step - loss: 0.4680 - accuracy: 0.7917 - val_loss: 0.485
2 - val_accuracy: 0.7832
Epoch 8/20
45994/45994 [==============================] - 76s 2ms/step - loss: 0.4563 - accuracy: 0.7971 - val_loss: 0.507
9 - val_accuracy: 0.7678
Epoch 9/20
45994/45994 [==============================] - 85s 2ms/step - loss: 0.4522 - accuracy: 0.7978 - val_loss: 0.484
7 - val_accuracy: 0.7837
Epoch 10/20
45994/45994 [==============================] - 71s 2ms/step - loss: 0.4446 - accuracy: 0.8031 - val_loss: 0.468
0 - val_accuracy: 0.7905
Epoch 11/20
45994/45994 [==============================] - 76s 2ms/step - loss: 0.4360 - accuracy: 0.8093 - val_loss: 0.479
4 - val_accuracy: 0.7873
Epoch 12/20
45994/45994 [==============================] - 84s 2ms/step - loss: 0.4314 - accuracy: 0.8100 - val_loss: 0.462
4 - val_accuracy: 0.7927
Epoch 13/20
45994/45994 [==============================] - 75s 2ms/step - loss: 0.4213 - accuracy: 0.8153 - val_loss: 0.473
8 - val_accuracy: 0.7847
Epoch 14/20
45994/45994 [==============================] - 77s 2ms/step - loss: 0.4174 - accuracy: 0.8196 - val_loss: 0.462
2 - val_accuracy: 0.7945
Epoch 15/20
45994/45994 [==============================] - 84s 2ms/step - loss: 0.4101 - accuracy: 0.8207 - val_loss: 0.494
4 - val_accuracy: 0.7791
Epoch 16/20
45994/45994 [==============================] - 79s 2ms/step - loss: 0.4045 - accuracy: 0.8227 - val_loss: 0.543
3 - val_accuracy: 0.7486
Epoch 17/20
45994/45994 [==============================] - 71s 2ms/step - loss: 0.4015 - accuracy: 0.8269 - val_loss: 0.507
9 - val_accuracy: 0.7765
Epoch 18/20
45994/45994 [==============================] - 67s 1ms/step - loss: 0.3948 - accuracy: 0.8289 - val_loss: 0.489
2 - val_accuracy: 0.7848
Epoch 19/20
45994/45994 [==============================] - 76s 2ms/step - loss: 0.3898 - accuracy: 0.8298 - val_loss: 0.487
7 - val_accuracy: 0.7864
Epoch 20/20
45994/45994 [==============================] - 72s 2ms/step - loss: 0.3813 - accuracy: 0.8360 - val_loss: 0.482
9 - val_accuracy: 0.7862
```

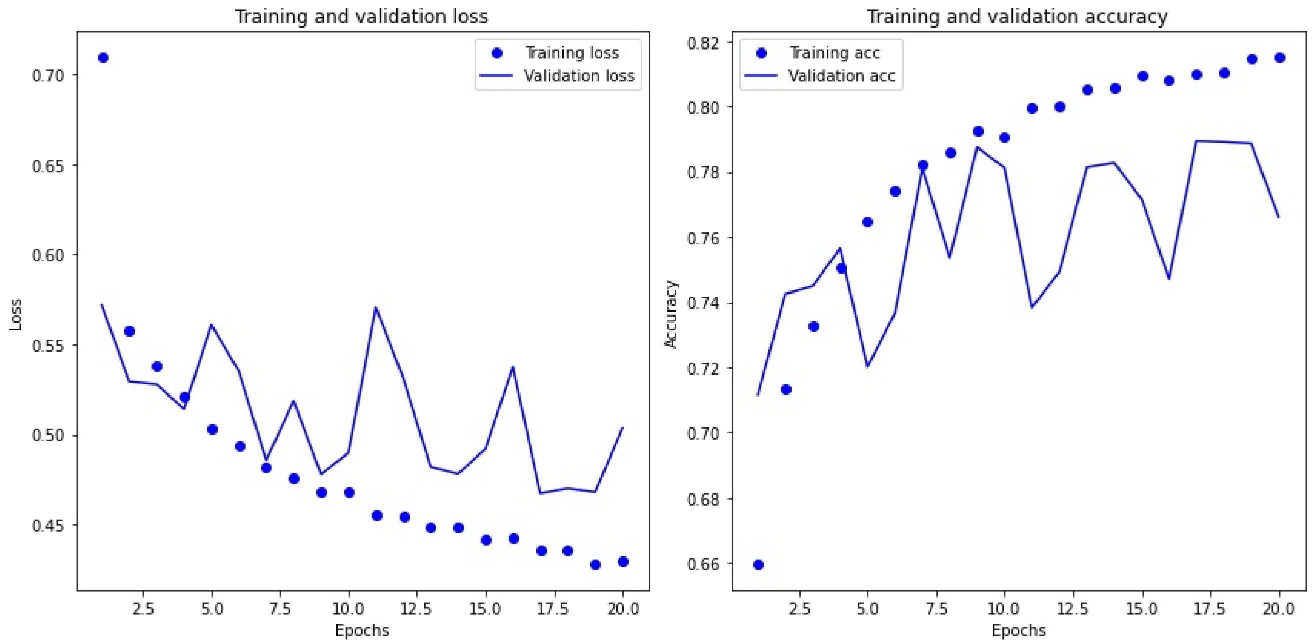


6.3 Model 3: Making the layers bigger (64 units)

In [13]:

```
uints = 64
learning_rate = 0.001

model_3 = create_model(units, learning_rate, add_layers)
m3_history = history(model_3, 20, 512)

m3_history_dict = m3_history
m3_loss_values = m3_history_dict['loss']
m3_val_loss_values = m3_history_dict['val_loss']
m3_acc_values = m3_history_dict['accuracy']
m3_val_acc_values = m3_history_dict['val_accuracy']

plot_history(m3_history_dict, m3_loss_values, m3_val_loss_values, m3_acc_values, m3_val_acc_values)
```

```
Train on 45994 samples, validate on 10000 samples
Epoch 1/20
45994/45994 [==============================] - 73s 2ms/step - loss: 0.7097 - accuracy: 0.6596 - val_loss: 0.571
7 - val_accuracy: 0.7116
Epoch 2/20
45994/45994 [==============================] - 68s 1ms/step - loss: 0.5574 - accuracy: 0.7135 - val_loss: 0.529
4 - val_accuracy: 0.7425
Epoch 3/20
45994/45994 [==============================] - 67s 1ms/step - loss: 0.5378 - accuracy: 0.7328 - val_loss: 0.527
7 - val_accuracy: 0.7449
Epoch 4/20
45994/45994 [==============================] - 61s 1ms/step - loss: 0.5209 - accuracy: 0.7504 - val_loss: 0.513
9 - val_accuracy: 0.7565
Epoch 5/20
45994/45994 [==============================] - 69s 2ms/step - loss: 0.5032 - accuracy: 0.7648 - val_loss: 0.560
8 - val_accuracy: 0.7201
Epoch 6/20
45994/45994 [==============================] - 68s 1ms/step - loss: 0.4938 - accuracy: 0.7743 - val_loss: 0.535
0 - val_accuracy: 0.7366
Epoch 7/20
45994/45994 [==============================] - 66s 1ms/step - loss: 0.4815 - accuracy: 0.7823 - val_loss: 0.485
5 - val_accuracy: 0.7811
Epoch 8/20
45994/45994 [==============================] - 67s 1ms/step - loss: 0.4760 - accuracy: 0.7858 - val_loss: 0.518
5 - val_accuracy: 0.7536
Epoch 9/20
45994/45994 [==============================] - 74s 2ms/step - loss: 0.4679 - accuracy: 0.7924 - val_loss: 0.477
7 - val_accuracy: 0.7875
Epoch 10/20
45994/45994 [==============================] - 62s 1ms/step - loss: 0.4679 - accuracy: 0.7905 - val_loss: 0.489
6 - val_accuracy: 0.7811
Epoch 11/20
45994/45994 [==============================] - 61s 1ms/step - loss: 0.4554 - accuracy: 0.7996 - val_loss: 0.570
5 - val_accuracy: 0.7383
Epoch 12/20
45994/45994 [==============================] - 61s 1ms/step - loss: 0.4545 - accuracy: 0.8003 - val_loss: 0.531
1 - val_accuracy: 0.7493
Epoch 13/20
45994/45994 [==============================] - 87s 2ms/step - loss: 0.4485 - accuracy: 0.8051 - val_loss: 0.481
8 - val_accuracy: 0.7813
Epoch 14/20
45994/45994 [==============================] - 78s 2ms/step - loss: 0.4482 - accuracy: 0.8056 - val_loss: 0.477
9 - val_accuracy: 0.7827
Epoch 15/20
45994/45994 [==============================] - 91s 2ms/step - loss: 0.4415 - accuracy: 0.8097 - val_loss: 0.491
8 - val_accuracy: 0.7716
Epoch 16/20
45994/45994 [==============================] - 91s 2ms/step - loss: 0.4420 - accuracy: 0.8080 - val_loss: 0.537
5 - val_accuracy: 0.7471
Epoch 17/20
45994/45994 [==============================] - 94s 2ms/step - loss: 0.4352 - accuracy: 0.8101 - val_loss: 0.467
1 - val_accuracy: 0.7894
Epoch 18/20
45994/45994 [==============================] - 73s 2ms/step - loss: 0.4359 - accuracy: 0.8106 - val_loss: 0.469
8 - val_accuracy: 0.7891
Epoch 19/20
45994/45994 [==============================] - 72s 2ms/step - loss: 0.4276 - accuracy: 0.8148 - val_loss: 0.467
8 - val_accuracy: 0.7886
Epoch 20/20
45994/45994 [==============================] - 81s 2ms/step - loss: 0.4293 - accuracy: 0.8152 - val_loss: 0.503
4 - val_accuracy: 0.7661
```



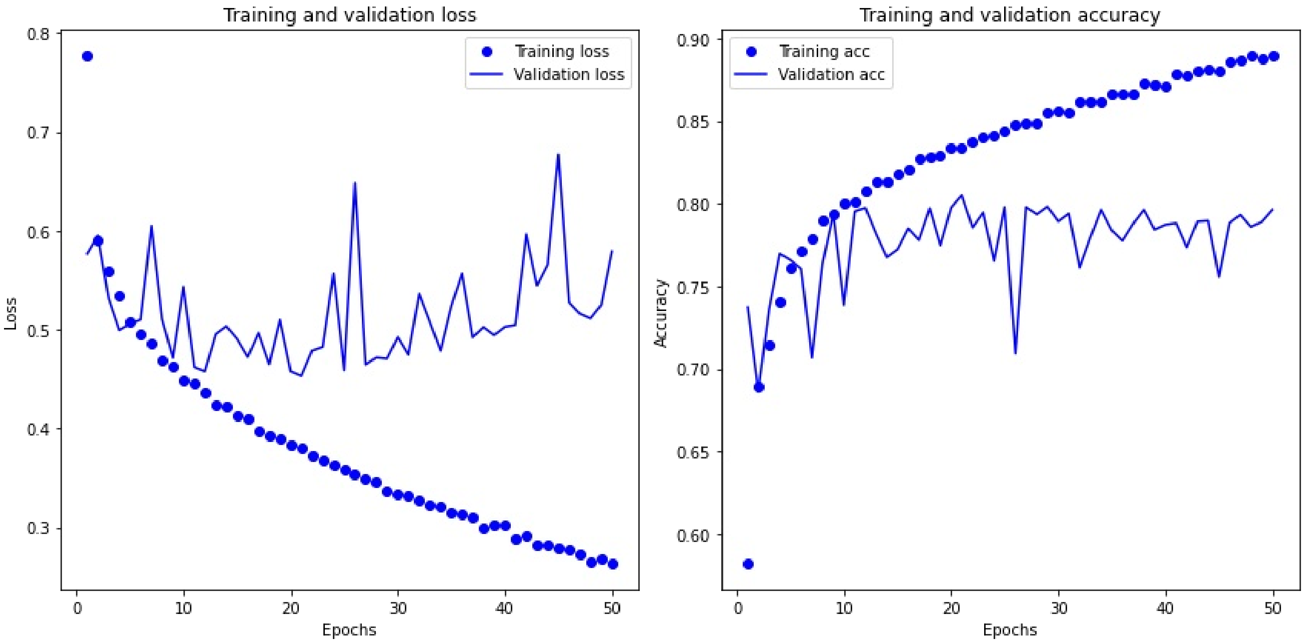
6.4 Model 4: Training for more epochs

In [14]:

```python
uints = 32
learning_rate = 0.001

model_4 = create_model(units, learning_rate, add_layers)
m4_history = history(model_4, 50, 512)

m4_history_dict = m4_history
m4_loss_values = m4_history_dict['loss']
m4_val_loss_values = m4_history_dict['val_loss']
m4_acc_values = m4_history_dict['accuracy']
m4_val_acc_values = m4_history_dict['val_accuracy']

plot_history(m4_history_dict, m4_loss_values, m4_val_loss_values, m4_acc_values, m4_val_acc_values)
```

```
Train on 45994 samples, validate on 10000 samples
Epoch 1/50
45994/45994 [==============================] - 93s 2ms/step - loss: 0.7772 - accuracy: 0.5821 - val_loss: 0.576
7 - val_accuracy: 0.7374
Epoch 2/50
45994/45994 [==============================] - 88s 2ms/step - loss: 0.5896 - accuracy: 0.6894 - val_loss: 0.595
1 - val_accuracy: 0.6861
Epoch 3/50
45994/45994 [==============================] - 104s 2ms/step - loss: 0.5586 - accuracy: 0.7150 - val_loss: 0.53
14 - val_accuracy: 0.7363
Epoch 4/50
45994/45994 [==============================] - 103s 2ms/step - loss: 0.5341 - accuracy: 0.7409 - val_loss: 0.49
93 - val_accuracy: 0.7697
Epoch 5/50
45994/45994 [==============================] - 113s 2ms/step - loss: 0.5072 - accuracy: 0.7610 - val_loss: 0.50
58 - val_accuracy: 0.7658
Epoch 6/50
45994/45994 [==============================] - 108s 2ms/step - loss: 0.4950 - accuracy: 0.7716 - val_loss: 0.51
00 - val_accuracy: 0.7606
Epoch 7/50
45994/45994 [==============================] - 95s 2ms/step - loss: 0.4855 - accuracy: 0.7792 - val_loss: 0.604
9 - val_accuracy: 0.7067
Epoch 8/50
45994/45994 [==============================] - 99s 2ms/step - loss: 0.4696 - accuracy: 0.7900 - val_loss: 0.510
5 - val_accuracy: 0.7637
Epoch 9/50
45994/45994 [==============================] - 91s 2ms/step - loss: 0.4635 - accuracy: 0.7940 - val_loss: 0.471
3 - val_accuracy: 0.7940
Epoch 10/50
45994/45994 [==============================] - 78s 2ms/step - loss: 0.4486 - accuracy: 0.8002 - val_loss: 0.543
4 - val_accuracy: 0.7387
Epoch 11/50
45994/45994 [==============================] - 79s 2ms/step - loss: 0.4457 - accuracy: 0.8014 - val_loss: 0.462
0 - val_accuracy: 0.7953
Epoch 12/50
45994/45994 [==============================] - 79s 2ms/step - loss: 0.4361 - accuracy: 0.8074 - val_loss: 0.457
7 - val_accuracy: 0.7974
Epoch 13/50
45994/45994 [==============================] - 82s 2ms/step - loss: 0.4235 - accuracy: 0.8134 - val_loss: 0.495
4 - val_accuracy: 0.7816
Epoch 14/50
45994/45994 [==============================] - 79s 2ms/step - loss: 0.4223 - accuracy: 0.8136 - val_loss: 0.503
4 - val_accuracy: 0.7676
Epoch 15/50
45994/45994 [==============================] - 80s 2ms/step - loss: 0.4135 - accuracy: 0.8178 - val_loss: 0.490
5 - val_accuracy: 0.7723
Epoch 16/50
45994/45994 [==============================] - 72s 2ms/step - loss: 0.4096 - accuracy: 0.8206 - val_loss: 0.472
3 - val_accuracy: 0.7849
Epoch 17/50
45994/45994 [==============================] - 99s 2ms/step - loss: 0.3981 - accuracy: 0.8268 - val_loss: 0.496
7 - val_accuracy: 0.7779
Epoch 18/50
45994/45994 [==============================] - 90s 2ms/step - loss: 0.3925 - accuracy: 0.8283 - val_loss: 0.464
9 - val_accuracy: 0.7972
Epoch 19/50
45994/45994 [==============================] - 98s 2ms/step - loss: 0.3897 - accuracy: 0.8289 - val_loss: 0.509
9 - val_accuracy: 0.7747
Epoch 20/50
45994/45994 [==============================] - 88s 2ms/step - loss: 0.3832 - accuracy: 0.8337 - val_loss: 0.457
7 - val_accuracy: 0.7977
Epoch 21/50
45994/45994 [==============================] - 102s 2ms/step - loss: 0.3802 - accuracy: 0.8339 - val_loss: 0.45
32 - val_accuracy: 0.8051
Epoch 22/50
45994/45994 [==============================] - 106s 2ms/step - loss: 0.3722 - accuracy: 0.8371 - val_loss: 0.47
88 - val_accuracy: 0.7855
Epoch 23/50
45994/45994 [==============================] - 114s 2ms/step - loss: 0.3676 - accuracy: 0.8399 - val_loss: 0.48
25 - val_accuracy: 0.7946
Epoch 24/50
45994/45994 [==============================] - 91s 2ms/step - loss: 0.3635 - accuracy: 0.8409 - val_loss: 0.556
```

```
6 - val_accuracy: 0.7653
Epoch 25/50
45994/45994 [==============================] - 98s 2ms/step - loss: 0.3582 - accuracy: 0.8437 - val_loss: 0.458
8 - val_accuracy: 0.7979
Epoch 26/50
45994/45994 [==============================] - 107s 2ms/step - loss: 0.3547 - accuracy: 0.8475 - val_loss: 0.64
81 - val_accuracy: 0.7093
Epoch 27/50
45994/45994 [==============================] - 106s 2ms/step - loss: 0.3487 - accuracy: 0.8489 - val_loss: 0.46
46 - val_accuracy: 0.7979
Epoch 28/50
45994/45994 [==============================] - 93s 2ms/step - loss: 0.3460 - accuracy: 0.8490 - val_loss: 0.471
8 - val_accuracy: 0.7935
Epoch 29/50
45994/45994 [==============================] - 90s 2ms/step - loss: 0.3371 - accuracy: 0.8547 - val_loss: 0.470
6 - val_accuracy: 0.7982
Epoch 30/50
45994/45994 [==============================] - 99s 2ms/step - loss: 0.3344 - accuracy: 0.8557 - val_loss: 0.492
5 - val_accuracy: 0.7892
Epoch 31/50
45994/45994 [==============================] - 104s 2ms/step - loss: 0.3325 - accuracy: 0.8554 - val_loss: 0.47
46 - val_accuracy: 0.7940
Epoch 32/50
45994/45994 [==============================] - 91s 2ms/step - loss: 0.3278 - accuracy: 0.8612 - val_loss: 0.536
2 - val_accuracy: 0.7613
Epoch 33/50
45994/45994 [==============================] - 85s 2ms/step - loss: 0.3226 - accuracy: 0.8618 - val_loss: 0.507
8 - val_accuracy: 0.7794
Epoch 34/50
45994/45994 [==============================] - 93s 2ms/step - loss: 0.3219 - accuracy: 0.8612 - val_loss: 0.478
6 - val_accuracy: 0.7963
Epoch 35/50
45994/45994 [==============================] - 97s 2ms/step - loss: 0.3149 - accuracy: 0.8661 - val_loss: 0.523
4 - val_accuracy: 0.7838
Epoch 36/50
45994/45994 [==============================] - 104s 2ms/step - loss: 0.3128 - accuracy: 0.8659 - val_loss: 0.55
68 - val_accuracy: 0.7775
Epoch 37/50
45994/45994 [==============================] - 88s 2ms/step - loss: 0.3109 - accuracy: 0.8659 - val_loss: 0.492
2 - val_accuracy: 0.7878
Epoch 38/50
45994/45994 [==============================] - 85s 2ms/step - loss: 0.2999 - accuracy: 0.8726 - val_loss: 0.502
5 - val_accuracy: 0.7963
Epoch 39/50
45994/45994 [==============================] - 82s 2ms/step - loss: 0.3020 - accuracy: 0.8721 - val_loss: 0.494
4 - val_accuracy: 0.7843
Epoch 40/50
45994/45994 [==============================] - 77s 2ms/step - loss: 0.3029 - accuracy: 0.8711 - val_loss: 0.502
7 - val_accuracy: 0.7872
Epoch 41/50
45994/45994 [==============================] - 93s 2ms/step - loss: 0.2884 - accuracy: 0.8785 - val_loss: 0.504
6 - val_accuracy: 0.7882
Epoch 42/50
45994/45994 [==============================] - 89s 2ms/step - loss: 0.2914 - accuracy: 0.8771 - val_loss: 0.596
5 - val_accuracy: 0.7735
Epoch 43/50
45994/45994 [==============================] - 82s 2ms/step - loss: 0.2830 - accuracy: 0.8800 - val_loss: 0.544
5 - val_accuracy: 0.7892
Epoch 44/50
45994/45994 [==============================] - 85s 2ms/step - loss: 0.2831 - accuracy: 0.8808 - val_loss: 0.565
8 - val_accuracy: 0.7897
Epoch 45/50
45994/45994 [==============================] - 83s 2ms/step - loss: 0.2793 - accuracy: 0.8804 - val_loss: 0.677
0 - val_accuracy: 0.7556
Epoch 46/50
45994/45994 [==============================] - 73s 2ms/step - loss: 0.2779 - accuracy: 0.8856 - val_loss: 0.527
5 - val_accuracy: 0.7883
Epoch 47/50
45994/45994 [==============================] - 82s 2ms/step - loss: 0.2727 - accuracy: 0.8864 - val_loss: 0.516
0 - val_accuracy: 0.7932
Epoch 48/50
45994/45994 [==============================] - 73s 2ms/step - loss: 0.2646 - accuracy: 0.8896 - val_loss: 0.511
2 - val_accuracy: 0.7859
Epoch 49/50
45994/45994 [==============================] - 84s 2ms/step - loss: 0.2689 - accuracy: 0.8874 - val_loss: 0.524
9 - val_accuracy: 0.7890
Epoch 50/50
45994/45994 [==============================] - 84s 2ms/step - loss: 0.2632 - accuracy: 0.8896 - val_loss: 0.579
0 - val_accuracy: 0.7963
```



7. Regularizing model and tuning hyperparameters

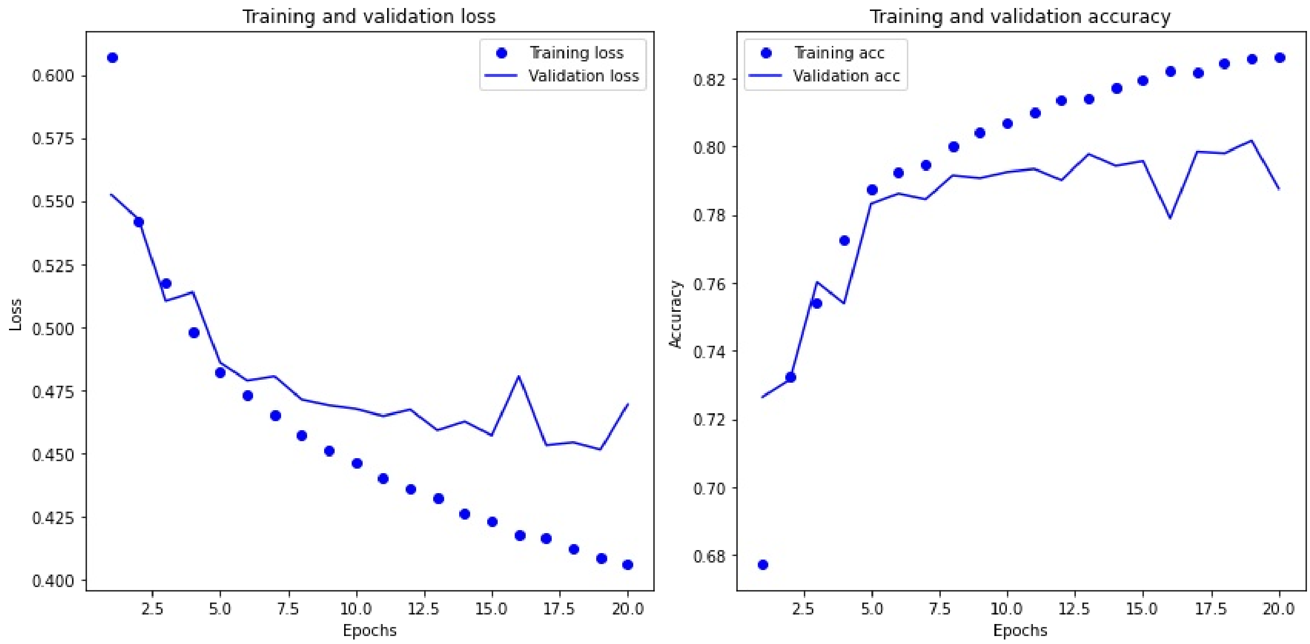
In pursuit of optimizing the model's performance, rigorous tuning of hyperparameters was undertaken. Initially, the learning rate was strategically decreased to mitigate the validation loss and accuracy fluctuations that were observed when the model overshot optimal points. This adjustment significantly enhanced the network, resulting in a more consistent validation accuracy and loss, as visually depicted in the accompanying plot. Then, the introduction of l1 regularization was implemented, contributing to a slight overfitting in the validation loss. Nevertheless, this regularization technique notably improved the overall accuracy of the model. After, the incorporation of l2 regularization was explored, which, while preventing overfitting, led to a deterioration in model performance. Consequently, l1 regularization was chosen as the preferred regularization method.

Further experimentation involved the addition of dropout. A dropout of 0.2 resulted in a marginal decrease in model performance, whereas a dropout of 0.5 yielded a slight improvement, though not significantly. Ultimately, the decision was made to forgo the use of dropout altogether. To attain a benchmark accuracy of 80%, deemed acceptable for the project, the final model underwent additional training epochs. The combination of these hyperparameter adjustments yielded an optimized model for this specific project requirements.

7.1 Model 5: Decreasing the learning rate

In [15]:
```
units = 32
learning_rate = 0.0001

model_5 = create_model(units, learning_rate, add_layers)
m5_history = history(model_5, 20, 512)

m5_history_dict = m5_history
m5_loss_values = m5_history_dict['loss']
m5_val_loss_values = m5_history_dict['val_loss']
m5_acc_values = m5_history_dict['accuracy']
m5_val_acc_values = m5_history_dict['val_accuracy']

plot_history(m5_history_dict, m5_loss_values, m5_val_loss_values, m5_acc_values, m5_val_acc_values)
```

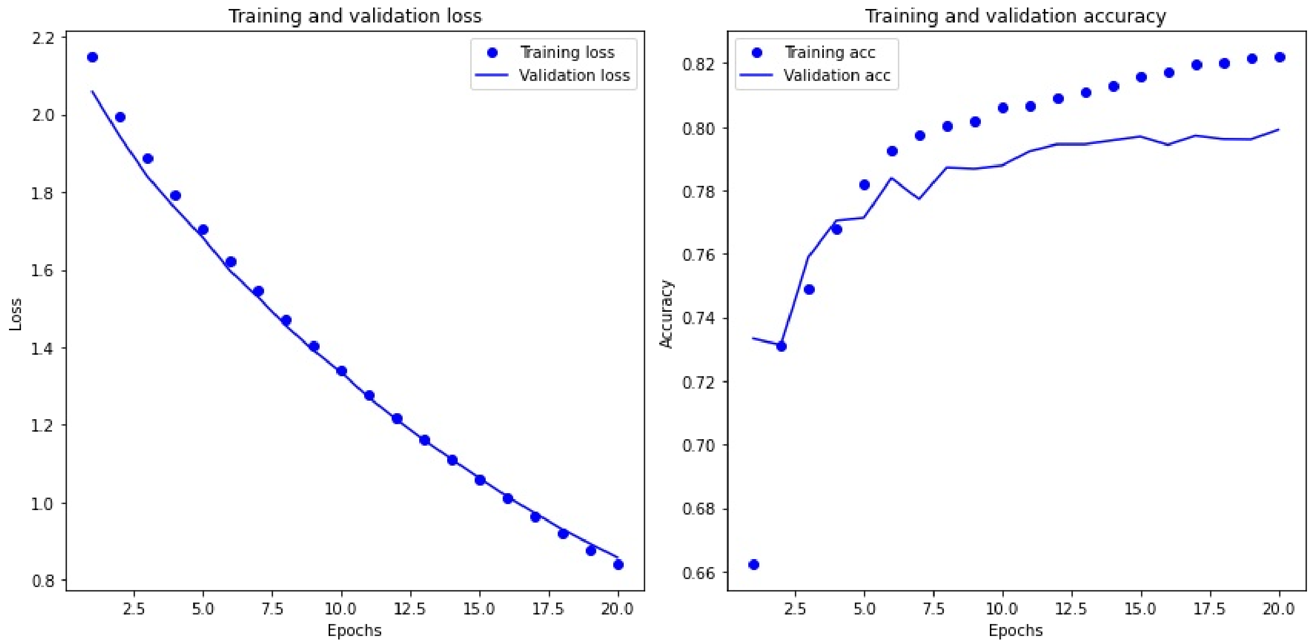
```
Train on 45994 samples, validate on 10000 samples
Epoch 1/20
45994/45994 [==============================] - 92s 2ms/step - loss: 0.6071 - accuracy: 0.6773 - val_loss: 0.552
4 - val_accuracy: 0.7265
Epoch 2/20
45994/45994 [==============================] - 78s 2ms/step - loss: 0.5421 - accuracy: 0.7326 - val_loss: 0.542
7 - val_accuracy: 0.7315
Epoch 3/20
45994/45994 [==============================] - 74s 2ms/step - loss: 0.5176 - accuracy: 0.7540 - val_loss: 0.510
3 - val_accuracy: 0.7602
Epoch 4/20
45994/45994 [==============================] - 71s 2ms/step - loss: 0.4983 - accuracy: 0.7726 - val_loss: 0.513
8 - val_accuracy: 0.7539
Epoch 5/20
45994/45994 [==============================] - 72s 2ms/step - loss: 0.4822 - accuracy: 0.7873 - val_loss: 0.485
9 - val_accuracy: 0.7832
Epoch 6/20
45994/45994 [==============================] - 70s 2ms/step - loss: 0.4730 - accuracy: 0.7925 - val_loss: 0.478
8 - val_accuracy: 0.7862
Epoch 7/20
45994/45994 [==============================] - 72s 2ms/step - loss: 0.4653 - accuracy: 0.7946 - val_loss: 0.480
5 - val_accuracy: 0.7845
Epoch 8/20
45994/45994 [==============================] - 75s 2ms/step - loss: 0.4574 - accuracy: 0.8002 - val_loss: 0.471
3 - val_accuracy: 0.7915
Epoch 9/20
45994/45994 [==============================] - 70s 2ms/step - loss: 0.4509 - accuracy: 0.8040 - val_loss: 0.469
0 - val_accuracy: 0.7907
Epoch 10/20
45994/45994 [==============================] - 4031s 88ms/step - loss: 0.4464 - accuracy: 0.8067 - val_loss: 0.
4676 - val_accuracy: 0.7925
Epoch 11/20
45994/45994 [==============================] - 1177s 26ms/step - loss: 0.4404 - accuracy: 0.8102 - val_loss: 0.
4647 - val_accuracy: 0.7934
Epoch 12/20
45994/45994 [==============================] - 93s 2ms/step - loss: 0.4361 - accuracy: 0.8137 - val_loss: 0.467
3 - val_accuracy: 0.7901
Epoch 13/20
45994/45994 [==============================] - 71s 2ms/step - loss: 0.4325 - accuracy: 0.8142 - val_loss: 0.459
1 - val_accuracy: 0.7978
Epoch 14/20
45994/45994 [==============================] - 70s 2ms/step - loss: 0.4262 - accuracy: 0.8171 - val_loss: 0.462
6 - val_accuracy: 0.7943
Epoch 15/20
45994/45994 [==============================] - 69s 2ms/step - loss: 0.4232 - accuracy: 0.8194 - val_loss: 0.457
0 - val_accuracy: 0.7957
Epoch 16/20
45994/45994 [==============================] - 63s 1ms/step - loss: 0.4179 - accuracy: 0.8223 - val_loss: 0.480
6 - val_accuracy: 0.7789
Epoch 17/20
45994/45994 [==============================] - 67s 1ms/step - loss: 0.4163 - accuracy: 0.8217 - val_loss: 0.453
2 - val_accuracy: 0.7985
Epoch 18/20
45994/45994 [==============================] - 61s 1ms/step - loss: 0.4119 - accuracy: 0.8247 - val_loss: 0.454
3 - val_accuracy: 0.7980
Epoch 19/20
45994/45994 [==============================] - 67s 1ms/step - loss: 0.4087 - accuracy: 0.8259 - val_loss: 0.451
4 - val_accuracy: 0.8017
Epoch 20/20
45994/45994 [==============================] - 67s 1ms/step - loss: 0.4059 - accuracy: 0.8263 - val_loss: 0.469
3 - val_accuracy: 0.7876
```



7.2 Model 6: Adding l1 regularisation

In [10]:
```
units = 32
learning_rate = 0.0001
regularization = 'l1'
add_layers = 0

model_6 = create_model(units, learning_rate, add_layers, regularization)
m6_history = history(model_6, 20, 512)

m6_history_dict = m6_history
m6_loss_values = m6_history_dict['loss']
m6_val_loss_values = m6_history_dict['val_loss']
m6_acc_values = m6_history_dict['accuracy']
m6_val_acc_values = m6_history_dict['val_accuracy']

plot_history(m6_history_dict, m6_loss_values, m6_val_loss_values, m6_acc_values, m6_val_acc_values)
```

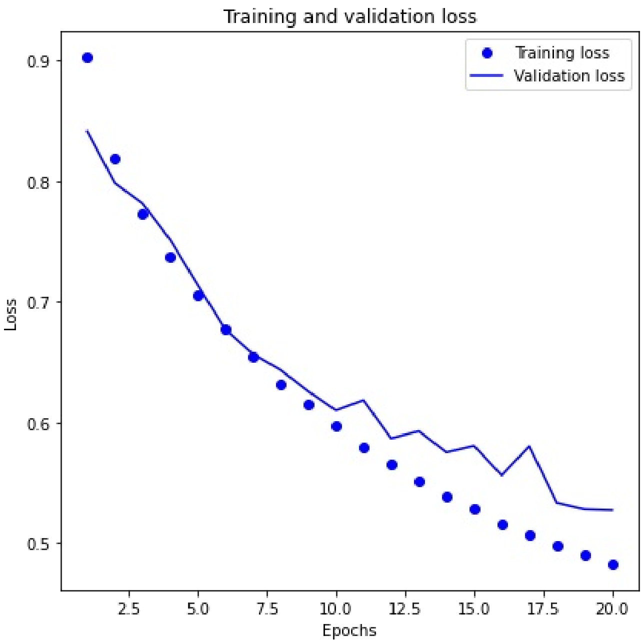
```
2024-01-14 20:28:05.518611: I tensorflow/core/platform/cpu_feature_guard.cc:145] This TensorFlow binary is opti
mized with Intel(R) MKL-DNN to use the following CPU instructions in performance critical operations:  SSE4.1 S
SE4.2
To enable them in non-MKL-DNN operations, rebuild TensorFlow with the appropriate compiler flags.
2024-01-14 20:28:05.528197: I tensorflow/core/common_runtime/process_util.cc:115] Creating new thread pool with
default inter op setting: 8. Tune using inter_op_parallelism_threads for best performance.
Train on 45994 samples, validate on 10000 samples
Epoch 1/20
45994/45994 [==============================] - 86s 2ms/step - loss: 2.1492 - accuracy: 0.6622 - val_loss: 2.058
1 - val_accuracy: 0.7333
Epoch 2/20
45994/45994 [==============================] - 117s 3ms/step - loss: 1.9944 - accuracy: 0.7309 - val_loss: 1.94
29 - val_accuracy: 0.7313
Epoch 3/20
45994/45994 [==============================] - 90s 2ms/step - loss: 1.8879 - accuracy: 0.7489 - val_loss: 1.839
1 - val_accuracy: 0.7590
Epoch 4/20
45994/45994 [==============================] - 86s 2ms/step - loss: 1.7916 - accuracy: 0.7679 - val_loss: 1.757
9 - val_accuracy: 0.7704
Epoch 5/20
45994/45994 [==============================] - 77s 2ms/step - loss: 1.7038 - accuracy: 0.7818 - val_loss: 1.682
4 - val_accuracy: 0.7713
Epoch 6/20
45994/45994 [==============================] - 76s 2ms/step - loss: 1.6211 - accuracy: 0.7924 - val_loss: 1.595
9 - val_accuracy: 0.7838
Epoch 7/20
45994/45994 [==============================] - 73s 2ms/step - loss: 1.5450 - accuracy: 0.7975 - val_loss: 1.528
6 - val_accuracy: 0.7772
Epoch 8/20
45994/45994 [==============================] - 80s 2ms/step - loss: 1.4721 - accuracy: 0.8002 - val_loss: 1.455
8 - val_accuracy: 0.7872
Epoch 9/20
45994/45994 [==============================] - 72s 2ms/step - loss: 1.4042 - accuracy: 0.8019 - val_loss: 1.391
8 - val_accuracy: 0.7868
Epoch 10/20
45994/45994 [==============================] - 71s 2ms/step - loss: 1.3394 - accuracy: 0.8061 - val_loss: 1.335
8 - val_accuracy: 0.7878
Epoch 11/20
45994/45994 [==============================] - 80s 2ms/step - loss: 1.2779 - accuracy: 0.8067 - val_loss: 1.270
6 - val_accuracy: 0.7923
Epoch 12/20
45994/45994 [==============================] - 70s 2ms/step - loss: 1.2189 - accuracy: 0.8089 - val_loss: 1.214
0 - val_accuracy: 0.7945
Epoch 13/20
45994/45994 [==============================] - 74s 2ms/step - loss: 1.1638 - accuracy: 0.8112 - val_loss: 1.160
9 - val_accuracy: 0.7945
Epoch 14/20
45994/45994 [==============================] - 82s 2ms/step - loss: 1.1096 - accuracy: 0.8129 - val_loss: 1.111
5 - val_accuracy: 0.7957
Epoch 15/20
45994/45994 [==============================] - 78s 2ms/step - loss: 1.0584 - accuracy: 0.8159 - val_loss: 1.063
0 - val_accuracy: 0.7969
Epoch 16/20
45994/45994 [==============================] - 90s 2ms/step - loss: 1.0098 - accuracy: 0.8174 - val_loss: 1.015
7 - val_accuracy: 0.7943
Epoch 17/20
45994/45994 [==============================] - 99s 2ms/step - loss: 0.9639 - accuracy: 0.8196 - val_loss: 0.973
0 - val_accuracy: 0.7972
Epoch 18/20
45994/45994 [==============================] - 91s 2ms/step - loss: 0.9194 - accuracy: 0.8203 - val_loss: 0.930
9 - val_accuracy: 0.7961
Epoch 19/20
45994/45994 [==============================] - 88s 2ms/step - loss: 0.8789 - accuracy: 0.8214 - val_loss: 0.894
0 - val_accuracy: 0.7960
Epoch 20/20
45994/45994 [==============================] - 95s 2ms/step - loss: 0.8402 - accuracy: 0.8222 - val_loss: 0.857
8 - val_accuracy: 0.7991
```

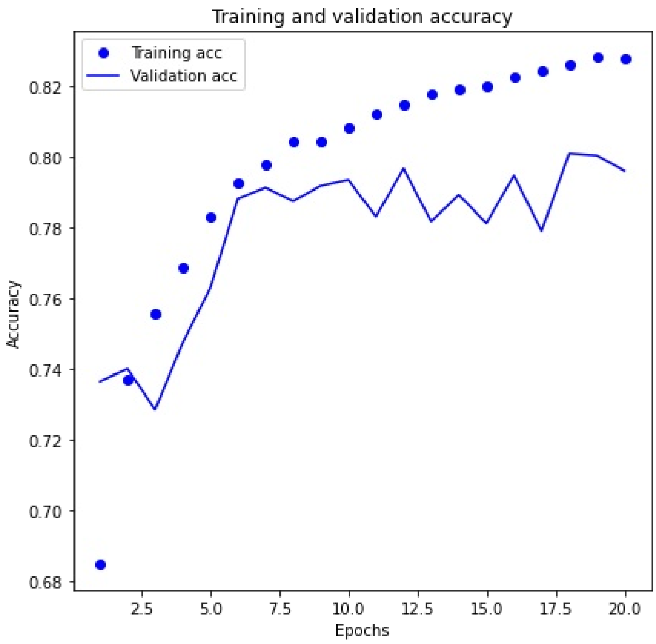


7.3 Model 7: Trying l2 regularisation

In [12]:
```
units = 32
learning_rate = 0.0001
regularization = 'l2'

model_7 = create_model(units, learning_rate, add_layers, regularization)
m7_history = history(model_7, 20, 512)

m7_history_dict = m7_history
m7_loss_values = m7_history_dict['loss']
m7_val_loss_values = m7_history_dict['val_loss']
m7_acc_values = m7_history_dict['accuracy']
m7_val_acc_values = m7_history_dict['val_accuracy']

plot_history(m7_history_dict, m7_loss_values, m7_val_loss_values, m7_acc_values, m7_val_acc_values)
```

```
Train on 45994 samples, validate on 10000 samples
Epoch 1/20
45994/45994 [==============================] - 89s 2ms/step - loss: 0.9031 - accuracy: 0.6847 - val_loss: 0.841
3 - val_accuracy: 0.7365
Epoch 2/20
45994/45994 [==============================] - 92s 2ms/step - loss: 0.8185 - accuracy: 0.7371 - val_loss: 0.798
4 - val_accuracy: 0.7401
Epoch 3/20
45994/45994 [==============================] - 79s 2ms/step - loss: 0.7734 - accuracy: 0.7559 - val_loss: 0.781
6 - val_accuracy: 0.7285
Epoch 4/20
45994/45994 [==============================] - 77s 2ms/step - loss: 0.7376 - accuracy: 0.7689 - val_loss: 0.751
5 - val_accuracy: 0.7475
Epoch 5/20
45994/45994 [==============================] - 87s 2ms/step - loss: 0.7059 - accuracy: 0.7832 - val_loss: 0.713
9 - val_accuracy: 0.7630
Epoch 6/20
45994/45994 [==============================] - 87s 2ms/step - loss: 0.6781 - accuracy: 0.7929 - val_loss: 0.677
0 - val_accuracy: 0.7883
Epoch 7/20
45994/45994 [==============================] - 87s 2ms/step - loss: 0.6544 - accuracy: 0.7978 - val_loss: 0.656
7 - val_accuracy: 0.7914
Epoch 8/20
45994/45994 [==============================] - 77s 2ms/step - loss: 0.6323 - accuracy: 0.8045 - val_loss: 0.643
5 - val_accuracy: 0.7876
Epoch 9/20
45994/45994 [==============================] - 95s 2ms/step - loss: 0.6145 - accuracy: 0.8046 - val_loss: 0.625
8 - val_accuracy: 0.7919
Epoch 10/20
45994/45994 [==============================] - 79s 2ms/step - loss: 0.5977 - accuracy: 0.8084 - val_loss: 0.610
0 - val_accuracy: 0.7936
Epoch 11/20
45994/45994 [==============================] - 100s 2ms/step - loss: 0.5799 - accuracy: 0.8121 - val_loss: 0.61
84 - val_accuracy: 0.7832
Epoch 12/20
45994/45994 [==============================] - 85s 2ms/step - loss: 0.5652 - accuracy: 0.8151 - val_loss: 0.586
7 - val_accuracy: 0.7968
Epoch 13/20
45994/45994 [==============================] - 95s 2ms/step - loss: 0.5513 - accuracy: 0.8178 - val_loss: 0.592
8 - val_accuracy: 0.7818
Epoch 14/20
45994/45994 [==============================] - 84s 2ms/step - loss: 0.5387 - accuracy: 0.8193 - val_loss: 0.575
2 - val_accuracy: 0.7894
Epoch 15/20
45994/45994 [==============================] - 91s 2ms/step - loss: 0.5279 - accuracy: 0.8203 - val_loss: 0.580
5 - val_accuracy: 0.7813
Epoch 16/20
45994/45994 [==============================] - 99s 2ms/step - loss: 0.5163 - accuracy: 0.8226 - val_loss: 0.556
2 - val_accuracy: 0.7949
Epoch 17/20
45994/45994 [==============================] - 86s 2ms/step - loss: 0.5068 - accuracy: 0.8245 - val_loss: 0.580
2 - val_accuracy: 0.7791
Epoch 18/20
45994/45994 [==============================] - 88s 2ms/step - loss: 0.4973 - accuracy: 0.8263 - val_loss: 0.533
3 - val_accuracy: 0.8011
Epoch 19/20
45994/45994 [==============================] - 84s 2ms/step - loss: 0.4900 - accuracy: 0.8285 - val_loss: 0.528
0 - val_accuracy: 0.8005
Epoch 20/20
45994/45994 [==============================] - 78s 2ms/step - loss: 0.4823 - accuracy: 0.8282 - val_loss: 0.527
3 - val_accuracy: 0.7962
```



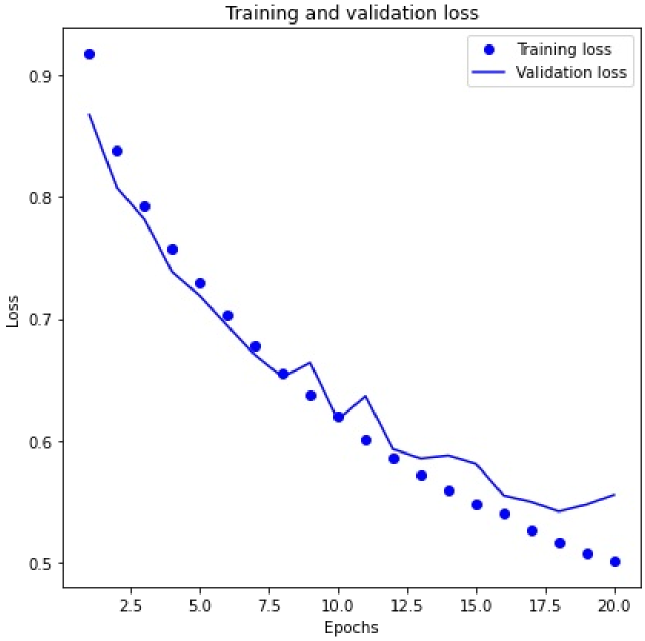


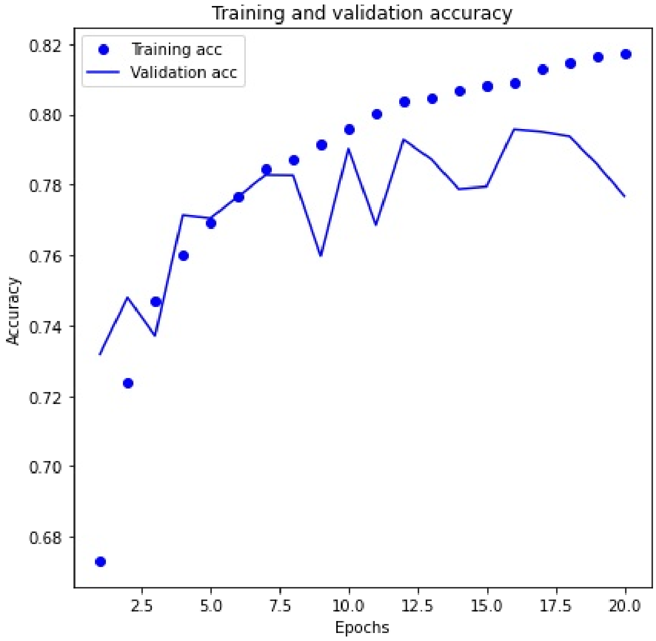
7.4 Model 8: Adding dropout of 0.2

In [18]:

```
units = 32
learning_rate = 0.0001
regularization = 'l2'
dropout = 0.2

model_8 = create_model(units, learning_rate, add_layers, regularization, dropout)
m8_history = history(model_8, 20, 512)

m8_history_dict = m8_history
m8_loss_values = m8_history_dict['loss']
m8_val_loss_values = m8_history_dict['val_loss']
m8_acc_values = m8_history_dict['accuracy']
m8_val_acc_values = m8_history_dict['val_accuracy']

plot_history(m8_history_dict, m8_loss_values, m8_val_loss_values, m8_acc_values, m8_val_acc_values)
```

```
Train on 45994 samples, validate on 10000 samples
Epoch 1/20
45994/45994 [==============================] - 119s 3ms/step - loss: 0.9175 - accuracy: 0.6730 - val_loss: 0.86
72 - val_accuracy: 0.7319
Epoch 2/20
45994/45994 [==============================] - 114s 2ms/step - loss: 0.8375 - accuracy: 0.7238 - val_loss: 0.80
78 - val_accuracy: 0.7480
Epoch 3/20
45994/45994 [==============================] - 112s 2ms/step - loss: 0.7928 - accuracy: 0.7471 - val_loss: 0.78
17 - val_accuracy: 0.7371
Epoch 4/20
45994/45994 [==============================] - 102s 2ms/step - loss: 0.7570 - accuracy: 0.7601 - val_loss: 0.73
86 - val_accuracy: 0.7714
Epoch 5/20
45994/45994 [==============================] - 120s 3ms/step - loss: 0.7297 - accuracy: 0.7693 - val_loss: 0.71
91 - val_accuracy: 0.7705
Epoch 6/20
45994/45994 [==============================] - 106s 2ms/step - loss: 0.7038 - accuracy: 0.7765 - val_loss: 0.69
45 - val_accuracy: 0.7766
Epoch 7/20
45994/45994 [==============================] - 144s 3ms/step - loss: 0.6785 - accuracy: 0.7847 - val_loss: 0.67
05 - val_accuracy: 0.7828
Epoch 8/20
45994/45994 [==============================] - 115s 3ms/step - loss: 0.6560 - accuracy: 0.7873 - val_loss: 0.65
23 - val_accuracy: 0.7827
Epoch 9/20
45994/45994 [==============================] - 96s 2ms/step - loss: 0.6378 - accuracy: 0.7917 - val_loss: 0.664
2 - val_accuracy: 0.7598
Epoch 10/20
45994/45994 [==============================] - 84s 2ms/step - loss: 0.6203 - accuracy: 0.7959 - val_loss: 0.617
5 - val_accuracy: 0.7903
Epoch 11/20
45994/45994 [==============================] - 87s 2ms/step - loss: 0.6010 - accuracy: 0.8001 - val_loss: 0.636
8 - val_accuracy: 0.7685
Epoch 12/20
45994/45994 [==============================] - 123s 3ms/step - loss: 0.5867 - accuracy: 0.8036 - val_loss: 0.59
34 - val_accuracy: 0.7928
Epoch 13/20
45994/45994 [==============================] - 119s 3ms/step - loss: 0.5723 - accuracy: 0.8045 - val_loss: 0.58
57 - val_accuracy: 0.7873
Epoch 14/20
45994/45994 [==============================] - 99s 2ms/step - loss: 0.5603 - accuracy: 0.8067 - val_loss: 0.588
2 - val_accuracy: 0.7788
Epoch 15/20
45994/45994 [==============================] - 98s 2ms/step - loss: 0.5489 - accuracy: 0.8080 - val_loss: 0.581
3 - val_accuracy: 0.7795
Epoch 16/20
45994/45994 [==============================] - 97s 2ms/step - loss: 0.5406 - accuracy: 0.8092 - val_loss: 0.555
4 - val_accuracy: 0.7958
Epoch 17/20
45994/45994 [==============================] - 146s 3ms/step - loss: 0.5271 - accuracy: 0.8128 - val_loss: 0.55
02 - val_accuracy: 0.7951
Epoch 18/20
45994/45994 [==============================] - 129s 3ms/step - loss: 0.5174 - accuracy: 0.8148 - val_loss: 0.54
24 - val_accuracy: 0.7938
Epoch 19/20
45994/45994 [==============================] - 104s 2ms/step - loss: 0.5082 - accuracy: 0.8165 - val_loss: 0.54
80 - val_accuracy: 0.7859
Epoch 20/20
45994/45994 [==============================] - 114s 2ms/step - loss: 0.5014 - accuracy: 0.8173 - val_loss: 0.55
60 - val_accuracy: 0.7768
```





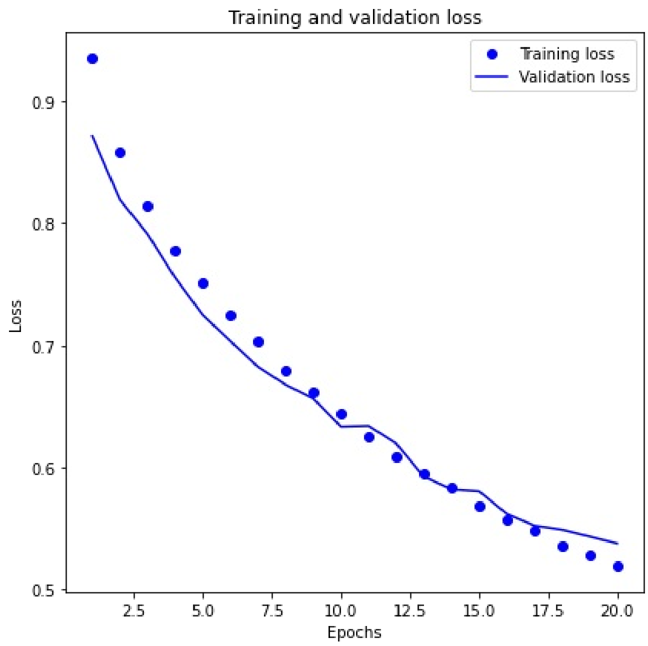
7.5 Model 9: Adding dropout of 0.5

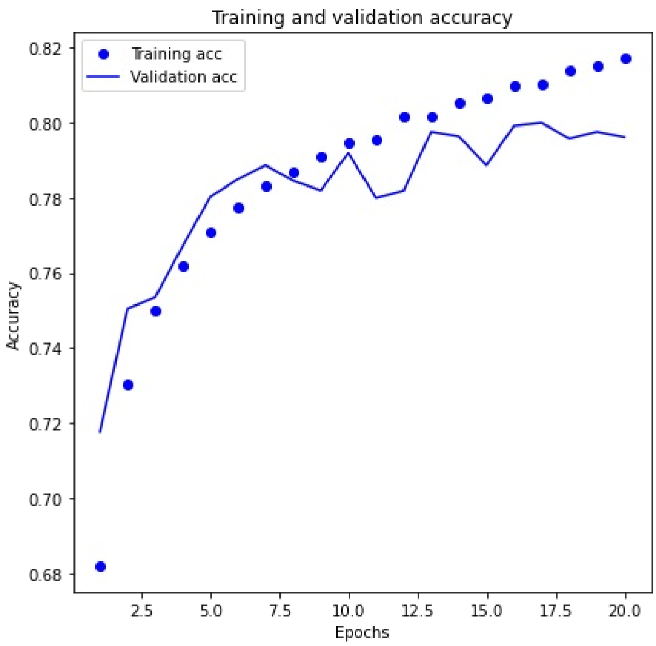
In [14]:

```
units = 32
learning_rate = 0.0001
regularization = 'l2'
dropout = 0.5

model_9 = create_model(units, learning_rate, add_layers, regularization, dropout)
m9_history = history(model_9, 20, 512)

m9_history_dict = m9_history
m9_loss_values = m9_history_dict['loss']
m9_val_loss_values = m9_history_dict['val_loss']
m9_acc_values = m9_history_dict['accuracy']
m9_val_acc_values = m9_history_dict['val_accuracy']

plot_history(m9_history_dict, m9_loss_values, m9_val_loss_values, m9_acc_values, m9_val_acc_values)
```

```
2024-01-14 23:15:28.824252: I tensorflow/core/platform/cpu_feature_guard.cc:145] This TensorFlow binary is opti
mized with Intel(R) MKL-DNN to use the following CPU instructions in performance critical operations:  SSE4.1 S
SE4.2
To enable them in non-MKL-DNN operations, rebuild TensorFlow with the appropriate compiler flags.
2024-01-14 23:15:28.828033: I tensorflow/core/common_runtime/process_util.cc:115] Creating new thread pool with
default inter op setting: 8. Tune using inter_op_parallelism_threads for best performance.
```

```
Train on 45994 samples, validate on 10000 samples
Epoch 1/20
45994/45994 [==============================] - 75s 2ms/step - loss: 0.9356 - accuracy: 0.6819 - val_loss: 0.871
5 - val_accuracy: 0.7177
Epoch 2/20
45994/45994 [==============================] - 90s 2ms/step - loss: 0.8585 - accuracy: 0.7304 - val_loss: 0.819
5 - val_accuracy: 0.7505
Epoch 3/20
45994/45994 [==============================] - 91s 2ms/step - loss: 0.8145 - accuracy: 0.7502 - val_loss: 0.791
4 - val_accuracy: 0.7535
Epoch 4/20
45994/45994 [==============================] - 77s 2ms/step - loss: 0.7784 - accuracy: 0.7621 - val_loss: 0.755
6 - val_accuracy: 0.7672
Epoch 5/20
45994/45994 [==============================] - 70s 2ms/step - loss: 0.7508 - accuracy: 0.7710 - val_loss: 0.725
0 - val_accuracy: 0.7803
Epoch 6/20
45994/45994 [==============================] - 75s 2ms/step - loss: 0.7255 - accuracy: 0.7774 - val_loss: 0.703
5 - val_accuracy: 0.7850
Epoch 7/20
45994/45994 [==============================] - 71s 2ms/step - loss: 0.7032 - accuracy: 0.7833 - val_loss: 0.682
5 - val_accuracy: 0.7887
Epoch 8/20
45994/45994 [==============================] - 74s 2ms/step - loss: 0.6791 - accuracy: 0.7870 - val_loss: 0.667
8 - val_accuracy: 0.7846
Epoch 9/20

45994/45994 [==============================] - 72s 2ms/step - loss: 0.6613 - accuracy: 0.7912 - val_loss: 0.656
5 - val_accuracy: 0.7820
Epoch 10/20
45994/45994 [==============================] - 66s 1ms/step - loss: 0.6439 - accuracy: 0.7947 - val_loss: 0.633
5 - val_accuracy: 0.7920
Epoch 11/20
45994/45994 [==============================] - 79s 2ms/step - loss: 0.6257 - accuracy: 0.7956 - val_loss: 0.634
1 - val_accuracy: 0.7800
Epoch 12/20
45994/45994 [==============================] - 58s 1ms/step - loss: 0.6090 - accuracy: 0.8015 - val_loss: 0.619
7 - val_accuracy: 0.7819
Epoch 13/20
45994/45994 [==============================] - 69s 2ms/step - loss: 0.5953 - accuracy: 0.8017 - val_loss: 0.592
6 - val_accuracy: 0.7976
Epoch 14/20
45994/45994 [==============================] - 81s 2ms/step - loss: 0.5838 - accuracy: 0.8053 - val_loss: 0.582
1 - val_accuracy: 0.7964
Epoch 15/20
45994/45994 [==============================] - 76s 2ms/step - loss: 0.5684 - accuracy: 0.8068 - val_loss: 0.580
6 - val_accuracy: 0.7887
Epoch 16/20
45994/45994 [==============================] - 87s 2ms/step - loss: 0.5571 - accuracy: 0.8098 - val_loss: 0.562
1 - val_accuracy: 0.7992
Epoch 17/20
45994/45994 [==============================] - 70s 2ms/step - loss: 0.5481 - accuracy: 0.8105 - val_loss: 0.552
4 - val_accuracy: 0.8000
Epoch 18/20
45994/45994 [==============================] - 63s 1ms/step - loss: 0.5359 - accuracy: 0.8140 - val_loss: 0.549
0 - val_accuracy: 0.7958
Epoch 19/20
45994/45994 [==============================] - 65s 1ms/step - loss: 0.5281 - accuracy: 0.8151 - val_loss: 0.543
5 - val_accuracy: 0.7976
Epoch 20/20
45994/45994 [==============================] - 66s 1ms/step - loss: 0.5190 - accuracy: 0.8173 - val_loss: 0.537
7 - val_accuracy: 0.7962
```



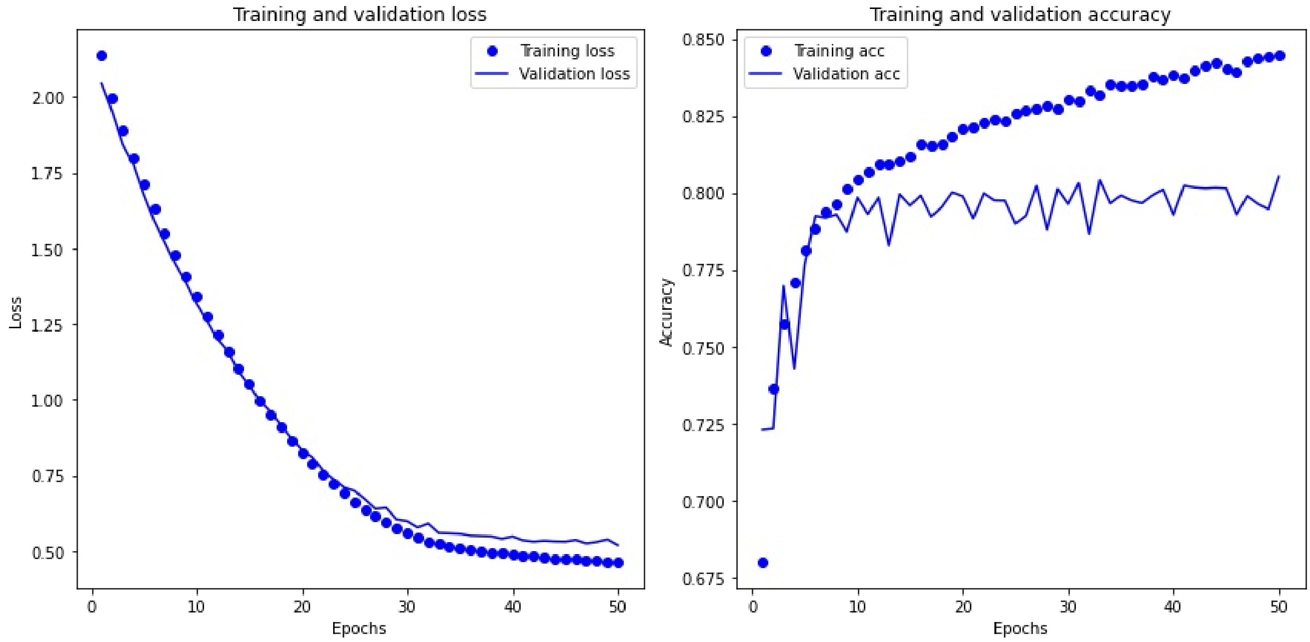


7.6 Model 10: Training Final Model on more Epochs

In [20]:
```
units = 32
learning_rate = 0.0001
regularization = 'l1'

model_10 = create_model(units, learning_rate, add_layers, regularization, dropout)
m10_history = history(model_10, 50, 512)

m10_history_dict = m10_history
m10_loss_values = m10_history_dict['loss']
m10_val_loss_values = m10_history_dict['val_loss']
m10_acc_values = m10_history_dict['accuracy']
m10_val_acc_values = m10_history_dict['val_accuracy']

plot_history(m10_history_dict, m10_loss_values, m10_val_loss_values, m10_acc_values, m10_val_acc_values)
```

```
Train on 45994 samples, validate on 10000 samples
Epoch 1/50
45994/45994 [==============================] - 99s 2ms/step - loss: 2.1391 - accuracy: 0.6798 - val_loss: 2.044
2 - val_accuracy: 0.7229
Epoch 2/50
45994/45994 [==============================] - 98s 2ms/step - loss: 1.9947 - accuracy: 0.7364 - val_loss: 1.952
3 - val_accuracy: 0.7233
Epoch 3/50
45994/45994 [==============================] - 91s 2ms/step - loss: 1.8903 - accuracy: 0.7574 - val_loss: 1.843
3 - val_accuracy: 0.7697
Epoch 4/50
45994/45994 [==============================] - 94s 2ms/step - loss: 1.7963 - accuracy: 0.7708 - val_loss: 1.778
4 - val_accuracy: 0.7427
Epoch 5/50
45994/45994 [==============================] - 86s 2ms/step - loss: 1.7103 - accuracy: 0.7811 - val_loss: 1.677
5 - val_accuracy: 0.7765
Epoch 6/50
45994/45994 [==============================] - 83s 2ms/step - loss: 1.6293 - accuracy: 0.7882 - val_loss: 1.589
5 - val_accuracy: 0.7924
Epoch 7/50
45994/45994 [==============================] - 95s 2ms/step - loss: 1.5523 - accuracy: 0.7938 - val_loss: 1.518
4 - val_accuracy: 0.7918
Epoch 8/50
45994/45994 [==============================] - 104s 2ms/step - loss: 1.4776 - accuracy: 0.7964 - val_loss: 1.44
70 - val_accuracy: 0.7929
Epoch 9/50
45994/45994 [==============================] - 95s 2ms/step - loss: 1.4076 - accuracy: 0.8013 - val_loss: 1.387
7 - val_accuracy: 0.7873
Epoch 10/50
45994/45994 [==============================] - 86s 2ms/step - loss: 1.3402 - accuracy: 0.8041 - val_loss: 1.317
2 - val_accuracy: 0.7984
Epoch 11/50
45994/45994 [==============================] - 83s 2ms/step - loss: 1.2775 - accuracy: 0.8066 - val_loss: 1.261
2 - val_accuracy: 0.7930
Epoch 12/50
45994/45994 [==============================] - 85s 2ms/step - loss: 1.2152 - accuracy: 0.8092 - val_loss: 1.199
7 - val_accuracy: 0.7984
Epoch 13/50
45994/45994 [==============================] - 92s 2ms/step - loss: 1.1570 - accuracy: 0.8091 - val_loss: 1.159
6 - val_accuracy: 0.7828
Epoch 14/50
45994/45994 [==============================] - 76s 2ms/step - loss: 1.1043 - accuracy: 0.8104 - val_loss: 1.093
0 - val_accuracy: 0.7995
Epoch 15/50
45994/45994 [==============================] - 83s 2ms/step - loss: 1.0507 - accuracy: 0.8119 - val_loss: 1.047
5 - val_accuracy: 0.7958
Epoch 16/50
45994/45994 [==============================] - 73s 2ms/step - loss: 0.9983 - accuracy: 0.8158 - val_loss: 0.997
3 - val_accuracy: 0.7991
Epoch 17/50
45994/45994 [==============================] - 89s 2ms/step - loss: 0.9529 - accuracy: 0.8155 - val_loss: 0.961
8 - val_accuracy: 0.7922
Epoch 18/50
45994/45994 [==============================] - 85s 2ms/step - loss: 0.9080 - accuracy: 0.8160 - val_loss: 0.916
5 - val_accuracy: 0.7953
Epoch 19/50
45994/45994 [==============================] - 103s 2ms/step - loss: 0.8651 - accuracy: 0.8182 - val_loss: 0.87
33 - val_accuracy: 0.8001
Epoch 20/50
45994/45994 [==============================] - 89s 2ms/step - loss: 0.8262 - accuracy: 0.8208 - val_loss: 0.833
9 - val_accuracy: 0.7988
Epoch 21/50
45994/45994 [==============================] - 87s 2ms/step - loss: 0.7874 - accuracy: 0.8213 - val_loss: 0.810
4 - val_accuracy: 0.7916
Epoch 22/50
45994/45994 [==============================] - 97s 2ms/step - loss: 0.7536 - accuracy: 0.8230 - val_loss: 0.766
2 - val_accuracy: 0.7998
Epoch 23/50
45994/45994 [==============================] - 88s 2ms/step - loss: 0.7216 - accuracy: 0.8237 - val_loss: 0.737
9 - val_accuracy: 0.7975
Epoch 24/50
```

```
45994/45994 [==============================] - 78s 2ms/step - loss: 0.6913 - accuracy: 0.8231 - val_loss: 0.710
8 - val_accuracy: 0.7974
Epoch 25/50
45994/45994 [==============================] - 86s 2ms/step - loss: 0.6634 - accuracy: 0.8259 - val_loss: 0.699
6 - val_accuracy: 0.7899
Epoch 26/50
45994/45994 [==============================] - 82s 2ms/step - loss: 0.6372 - accuracy: 0.8269 - val_loss: 0.671
4 - val_accuracy: 0.7925
Epoch 27/50
45994/45994 [==============================] - 99s 2ms/step - loss: 0.6150 - accuracy: 0.8272 - val_loss: 0.639
9 - val_accuracy: 0.8022
Epoch 28/50
45994/45994 [==============================] - 99s 2ms/step - loss: 0.5950 - accuracy: 0.8284 - val_loss: 0.643
7 - val_accuracy: 0.7880
Epoch 29/50
45994/45994 [==============================] - 105s 2ms/step - loss: 0.5769 - accuracy: 0.8275 - val_loss: 0.60
46 - val_accuracy: 0.8012
Epoch 30/50
45994/45994 [==============================] - 95s 2ms/step - loss: 0.5599 - accuracy: 0.8303 - val_loss: 0.598
9 - val_accuracy: 0.7963
Epoch 31/50
45994/45994 [==============================] - 93s 2ms/step - loss: 0.5444 - accuracy: 0.8298 - val_loss: 0.578
1 - val_accuracy: 0.8031
Epoch 32/50
45994/45994 [==============================] - 97s 2ms/step - loss: 0.5324 - accuracy: 0.8333 - val_loss: 0.591
2 - val_accuracy: 0.7866
Epoch 33/50
45994/45994 [==============================] - 90s 2ms/step - loss: 0.5241 - accuracy: 0.8320 - val_loss: 0.560
1 - val_accuracy: 0.8041
Epoch 34/50
45994/45994 [==============================] - 95s 2ms/step - loss: 0.5146 - accuracy: 0.8355 - val_loss: 0.558
7 - val_accuracy: 0.7965
Epoch 35/50
45994/45994 [==============================] - 79s 2ms/step - loss: 0.5097 - accuracy: 0.8350 - val_loss: 0.557
0 - val_accuracy: 0.7991
Epoch 36/50
45994/45994 [==============================] - 91s 2ms/step - loss: 0.5040 - accuracy: 0.8348 - val_loss: 0.550
9 - val_accuracy: 0.7975
Epoch 37/50
45994/45994 [==============================] - 87s 2ms/step - loss: 0.4999 - accuracy: 0.8355 - val_loss: 0.549
5 - val_accuracy: 0.7966
Epoch 38/50
45994/45994 [==============================] - 96s 2ms/step - loss: 0.4947 - accuracy: 0.8377 - val_loss: 0.548
3 - val_accuracy: 0.7991
Epoch 39/50
45994/45994 [==============================] - 95s 2ms/step - loss: 0.4927 - accuracy: 0.8367 - val_loss: 0.540
5 - val_accuracy: 0.8010
Epoch 40/50
45994/45994 [==============================] - 88s 2ms/step - loss: 0.4885 - accuracy: 0.8385 - val_loss: 0.548
2 - val_accuracy: 0.7928
Epoch 41/50
45994/45994 [==============================] - 101s 2ms/step - loss: 0.4862 - accuracy: 0.8372 - val_loss: 0.53
54 - val_accuracy: 0.8023
Epoch 42/50
45994/45994 [==============================] - 122s 3ms/step - loss: 0.4822 - accuracy: 0.8396 - val_loss: 0.53
09 - val_accuracy: 0.8017
Epoch 43/50
45994/45994 [==============================] - 120s 3ms/step - loss: 0.4788 - accuracy: 0.8414 - val_loss: 0.53
39 - val_accuracy: 0.8014
Epoch 44/50
45994/45994 [==============================] - 97s 2ms/step - loss: 0.4757 - accuracy: 0.8424 - val_loss: 0.531
7 - val_accuracy: 0.8016
Epoch 45/50
45994/45994 [==============================] - 115s 3ms/step - loss: 0.4728 - accuracy: 0.8402 - val_loss: 0.53
10 - val_accuracy: 0.8014
Epoch 46/50
45994/45994 [==============================] - 110s 2ms/step - loss: 0.4716 - accuracy: 0.8395 - val_loss: 0.53
66 - val_accuracy: 0.7929
Epoch 47/50
45994/45994 [==============================] - 107s 2ms/step - loss: 0.4685 - accuracy: 0.8429 - val_loss: 0.52
53 - val_accuracy: 0.7989
Epoch 48/50
45994/45994 [==============================] - 112s 2ms/step - loss: 0.4668 - accuracy: 0.8439 - val_loss: 0.52
99 - val_accuracy: 0.7964
Epoch 49/50
45994/45994 [==============================] - 107s 2ms/step - loss: 0.4640 - accuracy: 0.8444 - val_loss: 0.53
84 - val_accuracy: 0.7946
Epoch 50/50
45994/45994 [==============================] - 100s 2ms/step - loss: 0.4625 - accuracy: 0.8448 - val_loss: 0.51
94 - val_accuracy: 0.8052
```



In [21]:
```
# Summary of final model
model_10.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_10 (Dense)             (None, 32)                1881632   
_________________________________________________________________
dense_11 (Dense)             (None, 32)                1056      
_________________________________________________________________
dropout_4 (Dropout)          (None, 32)                0         
_________________________________________________________________
dense_12 (Dense)             (None, 1)                 33        
=================================================================
Total params: 1,882,721
Trainable params: 1,882,721
Non-trainable params: 0
_________________________________________________________________
```

---

8. Classifier evaluation

The classifier developed for this project demonstrates promising performance, surpassing a baseline accuracy of 50% with an achieved accuracy of 77%. An analysis of the learning curves indicates potential overfitting issues, as seen by oscillations in the validation loss and accuracy. To address this, strategic adjustments were made to hyperparameters. Although the accuracy only increased by 4% when compared to the baseline model, it achieved an accuracy of approximately 81% on the test set which is acceptable. This resistance to an increase in accuracy could indicate that more sophisticated deep learning approaches such as a convolutional neural network that works well with image data, need to be employed to adequately extract the patterns from the images.

In [27]:
```
# Evaluating the final model on the test set
results = model_10.evaluate(test_images, test_labels)
print(results[1])
```

```
13999/13999 [==============================] - 3s 179us/step
0.8093435168266296
```

---

9. Summary and Conclusion

9.1 Summary

In this study, a model was developed to classify galaxies as spiral or elliptical, aiming to surpass a baseline accuracy of 50%. The initial model, based on a template by Chollet, 2019, demonstrated an accuracy of 77%, exceeding the baseline. However, oscillations in validation loss and accuracy suggested potential overfitting due to a high learning rate. Subsequent experiments focused on addressing overfitting by adjusting network configurations and hyperparameters. The introduction of an additional layer did not improve accuracy, but increasing the number of units from 16 to 32 enhanced performance and reduced variations in validation metrics. Further experimentation with 64 units led to a decline in accuracy, prompting the decision to proceed with 32 units. Despite additional training epochs, significant accuracy improvement was not achieved, indicating potential limitations in the chosen model architecture.

To optimize performance, hyperparameter tuning ensued. The learning rate was strategically decreased, resulting in a more stable validation accuracy and loss. Subsequent implementation of l1 regularization, despite causing slight overfitting, improved overall accuracy. Conversely, l2 regularization led to a decline in performance, reinforcing the preference for l1 regularization. Dropout experiments showed minimal impact, leading to the decision to exclude dropout layers. The final model, refined through these adjustments, achieved a benchmark accuracy of 80%, considered acceptable for the project. However, analysis on the test set revealed an accuracy of approximately 81%, suggesting a resistance to further improvement with the current approach. This limitation may indicate the need for more sophisticated deep learning methods, such as convolutional neural networks tailored for image data, to extract intricate patterns from the images.

9.2 Conclusion

In conclusion, this developed classifier exhibits promising performance, surpassing the baseline and achieving satisfactory accuracy. The observed oscillations in validation metrics prompted systematic adjustments to mitigate overfitting, resulting in a more stable model. While the model achieved a desirable accuracy on the test set, further exploration of advanced deep learning approaches may be necessary to unlock the full potential of the data and improve accuracy beyond the current limitations of the model architecture.

---

10. References

1. Chris J. Lintott, Kevin Schawinski, Anže Slosar, Kate Land, Steven Bamford, Daniel Thomas, M. Jordan Raddick, Robert C. Nichol, Alex Szalay, Dan Andreescu, Phil Murray, Jan Vandenberg, Galaxy Zoo: morphologies derived from visual inspection of galaxies from the Sloan Digital Sky Survey, Monthly Notices of the Royal Astronomical Society, Volume 389, Issue 3, September 2008, Pages 1179–1189, https://doi.org/10.1111/j.1365-2966.2008.13689.x.
2. Sander Dieleman, Kyle W. Willett, Joni Dambre, Rotation-invariant convolutional neural networks for galaxy morphology prediction, Monthly Notices of the Royal Astronomical Society, Volume 450, Issue 2, 21 June 2015, Pages 1441–1459, https://doi.org/10.1093/mnras/stv632.
3. Gravet, Romaric & Cabrera-Vives, G. & Pérez-González, P. & Kartaltepe, J. & Barro, G. & Bernardi, M. & Mei, Simona & Shankar, F. & Dimauro, Paola & Bell, E. & Kocevski, D. & Koo, D. & Faber, Sabien & Mcintosh, D.. (2015). A Catalog of Visual-like Morphologies in the 5 CANDELS Fields Using Deep Learning. The Astrophysical Journal Supplement Series. 221. 10.1088/0067-0049/221/1/8.
4. P.H. Barchi, R.R. de Carvalho, R.R. Rosa, R.A. Sautter, M. Soares-Santos, B.A.D. Marques, E. Clua, T.S. Gonçalves, C. de Sá-Freitas, T.C. Moura, Machine and Deep Learning applied to galaxy morphology - A comparative study, Astronomy and Computing, Volume 30, 2020, 100334, ISSN 2213-1337, https://doi.org/10.1016/j.ascom.2019.100334.
5. Khan, A., Huerta, E.A., Wang, S., Gruendl, R., Jennings, E. and Zheng, H., 2019. Deep learning at scale for the construction of galaxy catalogs in the Dark Energy Survey. Physics Letters B, 795, pp.248-258.
6. Francois Chollet, Deep Learning Python, 2019.