lab06

May 23, 2024

0.0.1 People's Friendship University in Russia

Faculty of Science

Department of Mathematical Modeling and Artificial Intelligence

0.1 Labratory work №6 report

0.1.1 Meathods of machine learning

Student: Abu Suveilim Mukhammed M.

Group: NKNbd-01-21

0.2 Moscow 2024

0.2.1 Version №15

Option 15

1. Dataset oxford_iiit_pet with resolution changed to 60x96
2. Classes labeled 11,21,31,32,33
3. Requirements for MLP network architecture:

Serial API with add() method on creation

Loss Function: Categorical Cross Entropy

Number of hidden layers 6

The number of neurons is 30 in the first hidden layer, increasing by 15 with each subsequent hidden layer

Using layers with L1L2 regularization

4. CNN network architecture requirements:

Functional API when created

Loss Function: Sparse Categorical Cross-Entropy

Number of convolutional layers 2

Number of filters in convolutional layers 32

Filter dimensions 3x3

Using Batch Normalization Layers

5. Requirements for RNN network architecture:

Sequential API with list of layers on creation

Loss Function: Categorical Cross Entropy

LSTM layer with 96 neurons

Using dropout layers

6. Quality indicator of multi-class classification:

minimum class accuracy, where the accuracy of a class is equal to the proportion of correct predictions for all points assigned by the classifier to this class.

0.3 1. Load the data set with images specified in the individual task from Tensorflow Datasets, divided into training, validation and test samples. If during further work with the data there is a lack of computing resources, the image resolution can be reduced.

[1]:
```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf
import tensorflow_datasets as tfds
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
from matplotlib import rcParams
from mpl_toolkits.mplot3d import Axes3D
from PIL import Image, ImageOps
from math import sqrt
tf.__version__
```

[1]: '2.16.1'

[2]:
```python
# loading oxford_iiit_pet dataset
ds, info = tfds.load("oxford_iiit_pet", split=['train', 'test'], with_info=True)
print(ds)
print(info)
```

```
[<_PrefetchDataset element_spec={'file_name': TensorSpec(shape=(),
dtype=tf.string, name=None), 'image': TensorSpec(shape=(None, None, 3),
dtype=tf.uint8, name=None), 'label': TensorSpec(shape=(), dtype=tf.int64,
name=None), 'segmentation_mask': TensorSpec(shape=(None, None, 1),
dtype=tf.uint8, name=None), 'species': TensorSpec(shape=(), dtype=tf.int64,
name=None)}>, <_PrefetchDataset element_spec={'file_name': TensorSpec(shape=(),
dtype=tf.string, name=None), 'image': TensorSpec(shape=(None, None, 3),
dtype=tf.uint8, name=None), 'label': TensorSpec(shape=(), dtype=tf.int64,
```

```
name=None), 'segmentation_mask': TensorSpec(shape=(None, None, 1),
dtype=tf.uint8, name=None), 'species': TensorSpec(shape=(), dtype=tf.int64,
name=None)}>]
tfds.core.DatasetInfo(
    name='oxford_iiit_pet',
    full_name='oxford_iiit_pet/3.2.0',
    description="""
    The Oxford-IIIT pet dataset is a 37 category pet image dataset with roughly
200
    images for each class. The images have large variations in scale, pose and
    lighting. All images have an associated ground truth annotation of breed.
    """,
    homepage='http://www.robots.ox.ac.uk/~vgg/data/pets/',
    data_dir='C:\\Users\\Mo\\tensorflow_datasets\\oxford_iiit_pet\\3.2.0',
    file_format=tfrecord,
    download_size=773.52 MiB,
    dataset_size=774.69 MiB,
    features=FeaturesDict({
        'file_name': Text(shape=(), dtype=string),
        'image': Image(shape=(None, None, 3), dtype=uint8),
        'label': ClassLabel(shape=(), dtype=int64, num_classes=37),
        'segmentation_mask': Image(shape=(None, None, 1), dtype=uint8),
        'species': ClassLabel(shape=(), dtype=int64, num_classes=2),
    }),
    supervised_keys=('image', 'label'),
    disable_shuffling=False,
    splits={
        'test': <SplitInfo num_examples=3669, num_shards=4>,
        'train': <SplitInfo num_examples=3680, num_shards=4>,
    },
    citation="""@InProceedings{parkhi12a,
      author       = "Parkhi, O. M. and Vedaldi, A. and Zisserman, A. and
Jawahar, C.~V.",
      title        = "Cats and Dogs",
      booktitle    = "IEEE Conference on Computer Vision and Pattern
Recognition",
      year         = "2012",
    }""",
)
```

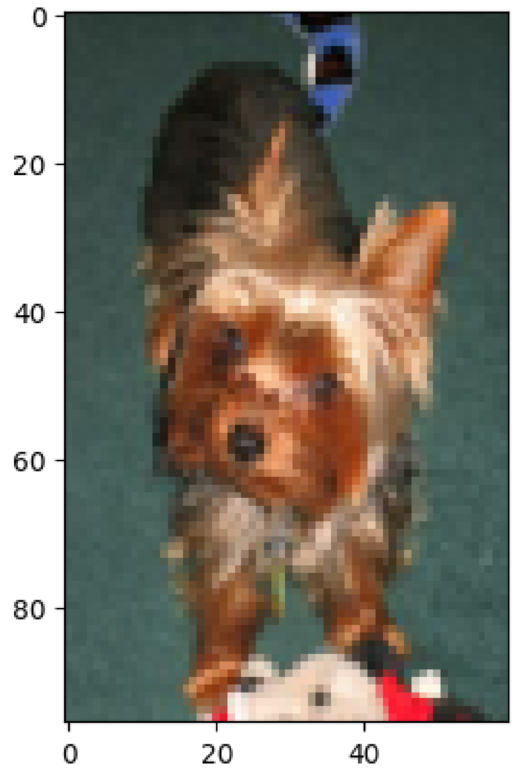
[3]:
```
df_train = tfds.as_dataframe(ds[0])
df_test  = tfds.as_dataframe(ds[1])
# Validation set (20% of df_train)
df_train, df_val = train_test_split(df_train, test_size=0.2, random_state=42)
print("training set:", df_train.shape, "validation set:", df_val.shape,␣
 ↪"testing set:", df_test.shape)
```

```
training set: (2944, 5) validation set: (736, 5) testing set: (3669, 5)
```

[4]:
```
image = Image.fromarray(df_train.iloc[0]['image'])
img = Image.fromarray(df_train.iloc[0]['image'])
image= image.resize((60,96))
```

[5]:
```
plt.imshow(image)
```

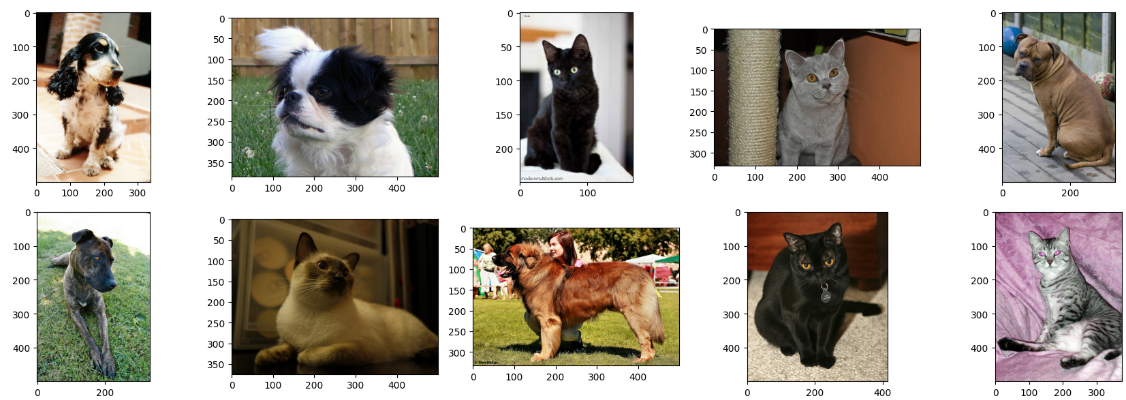
[5]: <matplotlib.image.AxesImage at 0x20566775a90>



[6]:
```
import random

def plot_random_sample(images):
    n = 10
    imgs = random.sample(list(images), n)

    num_row = 2
    num_col = 5

    fig, axes = plt.subplots(num_row, num_col, figsize=(3.5 * num_col, 3 *
 ↪num_row))
    # For every image
    for i in range(num_row * num_col):
        # Read the image
```

```
        img = imgs[i]
        # Display the image
        ax = axes[i // num_col, i % num_col]
        ax.imshow(img)

    plt.tight_layout()
    plt.show()
```

[7]:
```
plot_random_sample(df_train['image'])
```



[8]:
```
np.array(img).shape
```

[8]: (300, 225, 3)

[9]:
```
# Function to resize images in a DataFrame
def resize_images(df, new_size=(60, 96)):
    resized_images = []
    for i in range(df.shape[0]):
        image = Image.fromarray(df.iloc[i]['image'])
        image = image.resize(new_size)
        resized_images.append(np.array(image))
    df['image'] = resized_images
    return df

# Resize images in train and test DataFrames
df_train = resize_images(df_train)
df_test = resize_images(df_test)
df_val = resize_images(df_val)
```

[10]:
```
plot_random_sample(df_train['image'])
```

We can see that we changed the resolution to 60 by 96

[11]:
```
df_train.info
```

[11]:
```
<bound method DataFrame.info of                                               file_name  \
2618           b'yorkshire_terrier_147.jpg'
2964                b'newfoundland_148.jpg'
929             b'american_bulldog_100.jpg'
1837  b'staffordshire_bull_terrier_167.jpg'
1511                     b'Siamese_143.jpg'
…                                       …
1130                     b'Ragdoll_193.jpg'
1294                   b'chihuahua_178.jpg'
860              b'great_pyrenees_141.jpg'
3507                   b'shiba_inu_141.jpg'
3174                         b'pug_167.jpg'

                                                image  label  \
2618  [[[45, 60, 55], [48, 63, 58], [48, 63, 58], [4…     36
2964  [[[64, 36, 21], [53, 27, 14], [133, 110, 86], …     22
929   [[[29, 38, 30], [29, 44, 45], [27, 41, 44], [3…      1
1837  [[[255, 255, 255], [255, 255, 255], [248, 249,…     34
1511  [[[192, 204, 214], [199, 211, 224], [233, 239,…     32
…                                                 …    …
1130  [[[3, 3, 3], [2, 2, 2], [1, 1, 1], [2, 2, 2], …     26
1294  [[[42, 8, 0], [38, 8, 0], [34, 9, 0], [65, 20,…     10
860   [[[208, 191, 173], [145, 126, 109], [134, 116,…     15
3507  [[[128, 129, 127], [130, 130, 129], [128, 128,…     31
3174  [[[49, 78, 52], [56, 89, 69], [53, 83, 59], [5…     25

                                  segmentation_mask  species
2618  [[[2], [2], [2], [2], [2], [2], [2], [2], [2],…        1
```

```
2964  [[[2], [2], [2], [2], [2], [2], [2], [2], [2],…        1
929   [[[2], [2], [2], [2], [2], [2], [2], [2], [2],…        1
1837  [[[3], [3], [3], [3], [3], [3], [3], [3], [3],…        1
1511  [[[2], [2], [2], [2], [2], [2], [2], [2], [2],…        0
…                                                     …      …
1130  [[[2], [2], [2], [2], [2], [2], [2], [2], [2],…        0
1294  [[[2], [2], [2], [2], [2], [2], [2], [2], [2],…        1
860   [[[3], [3], [2], [2], [2], [2], [2], [2], [2],…        1
3507  [[[2], [2], [2], [2], [2], [2], [2], [2], [2],…        1
3174  [[[2], [2], [2], [2], [2], [2], [2], [2], [2],…        1

[2944 rows x 5 columns]>
```

[12]:
```
# Inspect unique label values in train, test and val DataFrames
print(df_train['label'].unique())
print(df_test['label'].unique())
print(df_val['label'].unique())
```

```
[36 22  1 34 32  4 25  9 14 19  5 16 12 11 30  6 10 33  2 29  8  7 23 17
 27 28 20 18 13 24 35 26 15 31 21  0  3]
[19 20 28  4 18 22 36 16  3 29 15 10 31  2  6  8  1 30 23 24 13 25 32 33
  7 21 17  9 34 12 14 26 27 11 35  0  5]
[14 29 35 16  1 18 33 13 22 32 27  5  4 36 26  6  8 20 11 23 28 30  9 34
 24 10  7  0  3 25 15  2 12 31 17 21 19]
```

0.4 2. Keep the images specified in the individual assignment in the set and render several images.

[13]:
```
# Filter and relabel DataFrames
def filter_and_relabel(df):
    x0 = df[df['label'] == 11]
    x0['label'] = 0
    x1 = df[df['label'] == 21]
    x1['label'] = 1
    x2 = df[df['label'] == 31]
    x2['label'] = 2
    x3 = df[df['label'] == 32]
    x3['label'] = 3
    x4 = df[df['label'] == 33]
    x4['label'] = 4
    return pd.concat([x0, x1, x2, x3, x4])

df_train_01 = filter_and_relabel(df_train)
df_val_01 = filter_and_relabel(df_val)
df_test_01 = filter_and_relabel(df_test)

print(df_train_01['label'].value_counts())
```

```python
print(df_val_01['label'].value_counts())
print(df_test_01['label'].value_counts())
```

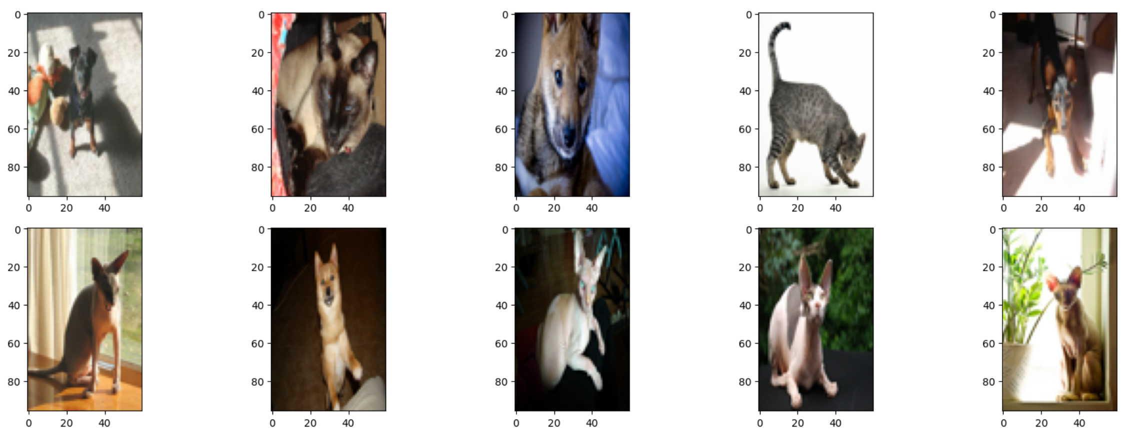
```
label
4    91
3    79
1    76
2    75
0    72
Name: count, dtype: int64
label
2    25
1    24
0    21
3    20
4     9
Name: count, dtype: int64
label
1    100
2    100
3    100
4    100
0     97
Name: count, dtype: int64
```

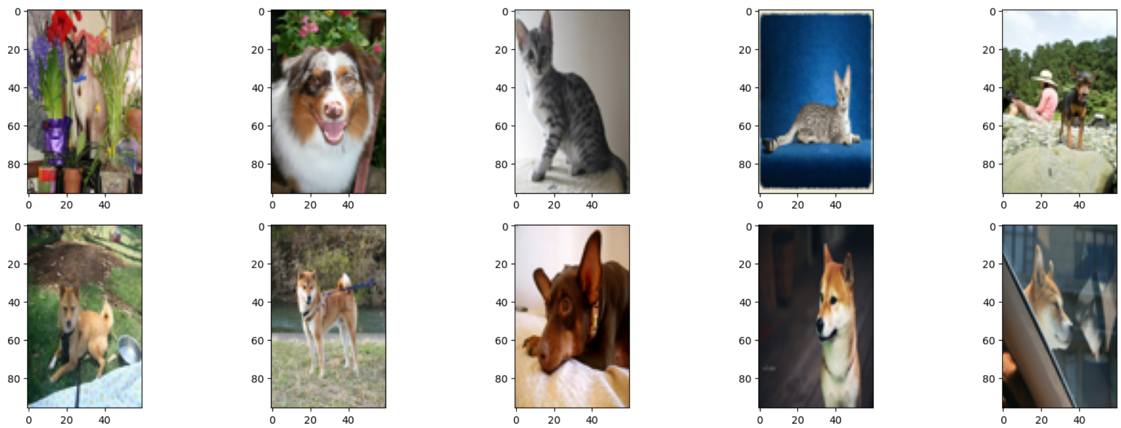
```
C:\Users\Mo\AppData\Local\Temp\ipykernel_11476\2111785736.py:4:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  x0['label'] = 0
C:\Users\Mo\AppData\Local\Temp\ipykernel_11476\2111785736.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  x1['label'] = 1
C:\Users\Mo\AppData\Local\Temp\ipykernel_11476\2111785736.py:8:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

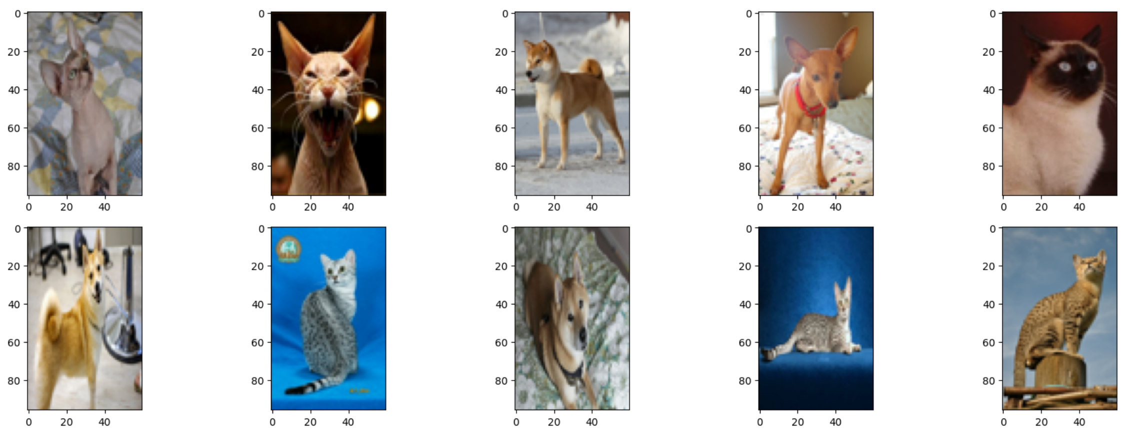
```
  x2['label'] = 2
C:\Users\Mo\AppData\Local\Temp\ipykernel_11476\2111785736.py:10:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  x3['label'] = 3
C:\Users\Mo\AppData\Local\Temp\ipykernel_11476\2111785736.py:12:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  x4['label'] = 4
```

[14]:

```
#let's check if we have correctly chose the lable
plot_random_sample(df_train_01['image'])
plot_random_sample(df_val_01['image'])
plot_random_sample(df_test_01['image'])
```







0.5 3. Build MLP, CNN and RNN neural networks for the task of multi-class image classification (network architecture requirements are specified in the individual task), using the loss function specified in the individual task. Select parameters such as activation functions, optimizer, initial learning rate, mini-batch size, etc. yourself, ensuring the training of neural networks. Train neural networks using the validation set generated in step 1. Stop training neural networks if losses on the validation set increase over several training epochs in a row. For each neural network, print the number of training epochs required.

0.5.1 MLP network

[15]:
```
train_labels_01 = df_train_01['label'].to_numpy(dtype=np.float32)
val_labels_01 = df_val_01['label'].to_numpy(dtype=np.float32)
test_labels_01 = df_test_01['label'].to_numpy(dtype=np.float32)
train_labels_01.shape, val_labels_01.shape, test_labels_01.shape
```

[15]: ((393,), (99,), (497,))

[16]:
```
label_train_01 = list(df_train_01['label'])
label_val_01 = list(df_val_01['label'])
label_test_01 = list(df_test_01['label'])
```

[17]:
```
def to_one_hot(labels, dimension=5):
    results = np.zeros((len(labels), dimension))
    for i, label in enumerate(labels):
        results[i, label] = 1.
    return results
```

[18]:
```
train_labels_01 = to_one_hot(label_train_01)
val_labels_01 = to_one_hot(label_val_01)
test_labels_01 = to_one_hot(label_test_01)
train_labels_01.shape, val_labels_01.shape, test_labels_01.shape
```

[18]: ((393, 5), (99, 5), (497, 5))

[19]:
```
train_images_01 = np.zeros(shape=(df_train_01.shape[0],96,60,3), dtype=np.
↪float32)
val_images_01 = np.zeros(shape=(df_val_01.shape[0],96,60,3), dtype=np.float32)
test_images_01 = np.zeros(shape=(df_test_01.shape[0],96,60,3), dtype=np.float32)
```

[20]:
```
for idx in range(train_labels_01.shape[0]):
    train_images_01[idx,:,:,:] = \
    np.array(Image.fromarray(df_train_01.iloc[idx]['image']))
for idx in range(test_labels_01.shape[0]):
    test_images_01[idx,:,:,:] = \
    np.array(Image.fromarray(df_test_01.iloc[idx]['image']))
```

[21]:
```
train_images_01 /= 255
val_images_01 /= 255
test_images_01 /= 255
print(train_images_01.shape, val_images_01.shape, test_images_01.shape,␣
↪train_labels_01.shape, val_labels_01.shape, test_labels_01.shape)
```

```
(393, 96, 60, 3) (99, 96, 60, 3) (497, 96, 60, 3) (393, 5) (99, 5) (497, 5)
```

Since we'll have 3 models I think it is better to make a function that creates each model So let's create a function for MLP networks with these parameters MLP network architecture requirements:

Serial API with add() method on creation

Loss Function: Categorical Cross Entropy

Number of hidden layers 6

The number of neurons is 30 in the first hidden layer, increasing by 15 with each subsequent hidden layer

Using layers with L1L2 regularization

[22]:
```
from tensorflow.keras import layers, regularizers, models
```

[23]:
```
def create_mlp_model(input_shape, num_classes):
    model = models.Sequential()
    model.add(layers.Flatten(input_shape=input_shape))
    model.add(layers.Dense(30, activation='swish', 
 ↪kernel_regularizer=regularizers.l1_l2(0.01)))
    #model.add(layers.Dropout(rate=0.5))
    model.add(layers.Dense(45, activation='swish', 
 ↪kernel_regularizer=regularizers.l1_l2(0.01)))
    #model.add(layers.Dropout(rate=0.5))
    model.add(layers.Dense(60, activation='swish', 
 ↪kernel_regularizer=regularizers.l1_l2(0.01)))
    #model.add(layers.Dropout(rate=0.5))
    model.add(layers.Dense(75, activation='swish', 
 ↪kernel_regularizer=regularizers.l1_l2(0.01)))
    #model.add(layers.Dropout(rate=0.5))
    model.add(layers.Dense(90, activation='swish', 
 ↪kernel_regularizer=regularizers.l1_l2(0.01)))
    #model.add(layers.Dropout(rate=0.5))
    model.add(layers.Dense(105, activation='swish', 
 ↪kernel_regularizer=regularizers.l1_l2(0.01)))
    model.add(layers.Dense(num_classes, activation='softmax'))
    return model

input_shape = (96, 60, 3)
num_classes = 5
```

[24]:
```
model_01 = create_mlp_model(input_shape, num_classes)
model_01.compile(optimizer='adam', loss='categorical_crossentropy', 
 ↪metrics=['accuracy'])
model_01.summary()
```

```
C:\Users\Mo\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kf
ra8p0\LocalCache\local-packages\Python311\site-
packages\keras\src\layers\reshaping\flatten.py:37: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

```
Model: "sequential"


 Layer (type)                           Output Shape                          
↪Param #
```

```
flatten (Flatten)                    (None, 17280)                          ␣
↪  0

dense (Dense)                        (None, 30)                          ␣
↪518,430

dense_1 (Dense)                      (None, 45)                           ␣
↪1,395

dense_2 (Dense)                      (None, 60)                           ␣
↪2,760

dense_3 (Dense)                      (None, 75)                           ␣
↪4,575

dense_4 (Dense)                      (None, 90)                           ␣
↪6,840

dense_5 (Dense)                      (None, 105)                          ␣
↪9,555

dense_6 (Dense)                      (None, 5)                             ␣
↪530
```

Total params: 544,085 (2.08 MB)

Trainable params: 544,085 (2.08 MB)

Non-trainable params: 0 (0.00 B)

[25]:
```
from tensorflow.keras.callbacks import EarlyStopping
# EarlyStopping callback
early_stopping = EarlyStopping(
    monitor="val_loss",
    min_delta=0,
    patience=10, #After 10 epochs with no improvement  training will be stopped
    verbose=1, #displays messages when the callback takes an action
    mode="auto",
    baseline=None,
    restore_best_weights=True, #restore best weights
    start_from_epoch=10, #warmup 10 epochs
)
```

[26]:
```python
# Train MLP model
history_01 = model_01.fit(
    train_images_01, train_labels_01,
    epochs=50, batch_size=128,
    validation_data=(val_images_01, val_labels_01),
    callbacks=[early_stopping]
)
```

```
Epoch 1/50
4/4              2s 57ms/step -
accuracy: 0.2273 - loss: 71.0410 - val_accuracy: 0.0909 - val_loss: 55.8369
Epoch 2/50
4/4              0s 8ms/step -
accuracy: 0.2293 - loss: 53.0722 - val_accuracy: 0.0909 - val_loss: 41.9381
Epoch 3/50
4/4              0s 8ms/step -
accuracy: 0.2314 - loss: 40.0536 - val_accuracy: 0.0909 - val_loss: 32.8126
Epoch 4/50
4/4              0s 8ms/step -
accuracy: 0.2338 - loss: 31.7476 - val_accuracy: 0.0909 - val_loss: 28.2876
Epoch 5/50
4/4              0s 7ms/step -
accuracy: 0.2278 - loss: 27.9945 - val_accuracy: 0.0909 - val_loss: 27.5801
Epoch 6/50
4/4              0s 7ms/step -
accuracy: 0.2171 - loss: 27.3229 - val_accuracy: 0.0909 - val_loss: 25.6368
Epoch 7/50
4/4              0s 8ms/step -
accuracy: 0.2247 - loss: 25.1421 - val_accuracy: 0.0909 - val_loss: 23.0816
Epoch 8/50
4/4              0s 8ms/step -
accuracy: 0.2148 - loss: 22.7095 - val_accuracy: 0.0909 - val_loss: 21.5340
Epoch 9/50
4/4              0s 7ms/step -
accuracy: 0.2265 - loss: 21.3653 - val_accuracy: 0.0909 - val_loss: 20.5221
Epoch 10/50
4/4              0s 8ms/step -
accuracy: 0.2275 - loss: 20.2328 - val_accuracy: 0.0909 - val_loss: 19.1098
Epoch 11/50
4/4              0s 10ms/step -
accuracy: 0.2338 - loss: 18.8970 - val_accuracy: 0.0909 - val_loss: 18.1095
Epoch 12/50
4/4              0s 8ms/step -
accuracy: 0.2338 - loss: 17.9231 - val_accuracy: 0.0909 - val_loss: 17.0831
Epoch 13/50
4/4              0s 8ms/step -
accuracy: 0.2278 - loss: 16.8877 - val_accuracy: 0.0909 - val_loss: 16.3111
Epoch 14/50
```

```
4/4            0s 8ms/step -
accuracy: 0.2306 - loss: 16.1237 - val_accuracy: 0.0909 - val_loss: 15.4013
Epoch 15/50
4/4            0s 8ms/step -
accuracy: 0.2450 - loss: 15.2541 - val_accuracy: 0.0909 - val_loss: 14.6904
Epoch 16/50
4/4            0s 8ms/step -
accuracy: 0.2293 - loss: 14.5181 - val_accuracy: 0.0909 - val_loss: 13.9604
Epoch 17/50
4/4            0s 8ms/step -
accuracy: 0.2353 - loss: 13.8117 - val_accuracy: 0.0909 - val_loss: 13.2497
Epoch 18/50
4/4            0s 9ms/step -
accuracy: 0.2411 - loss: 13.1049 - val_accuracy: 0.0909 - val_loss: 12.5992
Epoch 19/50
4/4            0s 9ms/step -
accuracy: 0.2262 - loss: 12.4592 - val_accuracy: 0.0909 - val_loss: 11.9486
Epoch 20/50
4/4            0s 8ms/step -
accuracy: 0.2411 - loss: 11.8206 - val_accuracy: 0.0909 - val_loss: 11.3280
Epoch 21/50
4/4            0s 8ms/step -
accuracy: 0.2239 - loss: 11.2023 - val_accuracy: 0.0909 - val_loss: 10.7538
Epoch 22/50
4/4            0s 8ms/step -
accuracy: 0.2491 - loss: 10.6273 - val_accuracy: 0.0909 - val_loss: 10.1968
Epoch 23/50
4/4            0s 7ms/step -
accuracy: 0.2288 - loss: 10.0774 - val_accuracy: 0.0909 - val_loss: 9.6345
Epoch 24/50
4/4            0s 7ms/step -
accuracy: 0.2387 - loss: 9.5241 - val_accuracy: 0.0909 - val_loss: 9.1103
Epoch 25/50
4/4            0s 8ms/step -
accuracy: 0.2312 - loss: 8.9974 - val_accuracy: 0.0909 - val_loss: 8.6045
Epoch 26/50
4/4            0s 8ms/step -
accuracy: 0.2335 - loss: 8.5017 - val_accuracy: 0.0909 - val_loss: 8.1225
Epoch 27/50
4/4            0s 8ms/step -
accuracy: 0.2335 - loss: 8.0131 - val_accuracy: 0.0909 - val_loss: 7.6770
Epoch 28/50
4/4            0s 8ms/step -
accuracy: 0.2374 - loss: 7.5699 - val_accuracy: 0.0909 - val_loss: 7.2329
Epoch 29/50
4/4            0s 8ms/step -
accuracy: 0.2231 - loss: 7.1426 - val_accuracy: 0.0909 - val_loss: 6.8103
Epoch 30/50
```

```
4/4            0s 8ms/step -
accuracy: 0.2301 - loss: 6.7159 - val_accuracy: 0.0909 - val_loss: 6.4183
Epoch 31/50
4/4            0s 7ms/step -
accuracy: 0.2309 - loss: 6.3248 - val_accuracy: 0.0909 - val_loss: 6.0416
Epoch 32/50
4/4            0s 8ms/step -
accuracy: 0.2312 - loss: 5.9553 - val_accuracy: 0.0909 - val_loss: 5.6841
Epoch 33/50
4/4            0s 8ms/step -
accuracy: 0.2309 - loss: 5.5976 - val_accuracy: 0.0909 - val_loss: 5.3612
Epoch 34/50
4/4            0s 8ms/step -
accuracy: 0.2385 - loss: 5.2775 - val_accuracy: 0.0909 - val_loss: 5.0385
Epoch 35/50
4/4            0s 8ms/step -
accuracy: 0.2455 - loss: 4.9626 - val_accuracy: 0.0909 - val_loss: 4.7250
Epoch 36/50
4/4            0s 8ms/step -
accuracy: 0.2278 - loss: 4.6565 - val_accuracy: 0.0909 - val_loss: 4.4719
Epoch 37/50
4/4            0s 8ms/step -
accuracy: 0.2194 - loss: 4.3943 - val_accuracy: 0.0909 - val_loss: 4.2185
Epoch 38/50
4/4            0s 7ms/step -
accuracy: 0.2280 - loss: 4.1535 - val_accuracy: 0.0909 - val_loss: 3.9782
Epoch 39/50
4/4            0s 8ms/step -
accuracy: 0.2317 - loss: 3.9147 - val_accuracy: 0.0909 - val_loss: 3.7828
Epoch 40/50
4/4            0s 8ms/step -
accuracy: 0.2319 - loss: 3.7179 - val_accuracy: 0.0909 - val_loss: 3.5758
Epoch 41/50
4/4            0s 8ms/step -
accuracy: 0.2439 - loss: 3.5218 - val_accuracy: 0.0909 - val_loss: 3.4060
Epoch 42/50
4/4            0s 19ms/step -
accuracy: 0.2332 - loss: 3.3490 - val_accuracy: 0.0909 - val_loss: 3.2624
Epoch 43/50
4/4            0s 9ms/step -
accuracy: 0.2166 - loss: 3.2122 - val_accuracy: 0.0909 - val_loss: 3.1402
Epoch 44/50
4/4            0s 8ms/step -
accuracy: 0.2338 - loss: 3.0933 - val_accuracy: 0.0909 - val_loss: 3.0168
Epoch 45/50
4/4            0s 8ms/step -
accuracy: 0.2254 - loss: 2.9752 - val_accuracy: 0.0909 - val_loss: 2.9214
Epoch 46/50
```

```
4/4              0s 8ms/step -
accuracy: 0.2348 - loss: 2.8753 - val_accuracy: 0.0909 - val_loss: 2.8331
Epoch 47/50
4/4              0s 8ms/step -
accuracy: 0.2301 - loss: 2.7924 - val_accuracy: 0.0909 - val_loss: 2.7483
Epoch 48/50
4/4              0s 8ms/step -
accuracy: 0.2301 - loss: 2.7091 - val_accuracy: 0.0909 - val_loss: 2.7046
Epoch 49/50
4/4              0s 8ms/step -
accuracy: 0.2244 - loss: 2.6652 - val_accuracy: 0.0909 - val_loss: 2.6350
Epoch 50/50
4/4              0s 8ms/step -
accuracy: 0.2455 - loss: 2.5983 - val_accuracy: 0.0909 - val_loss: 2.5777
Restoring model weights from the end of the best epoch: 50.
```

0.5.2 CNN network

[27]:
```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D as MaxPool2D,␣
↪Flatten, Dense, BatchNormalization
```

[28]:
```
def create_cnn_model(input_shape, num_classes):
    layer1 = Input(shape = input_shape)
    layer2 = Conv2D(filters=32, kernel_size=(3, 3), input_shape=input_shape,␣
↪activation='leaky_relu')(layer1)
    layer3 = BatchNormalization()(layer2)
    layer4 = MaxPool2D(pool_size=(3, 3), padding='same')(layer3)
    layer5 = BatchNormalization()(layer4)
    layer6 = Conv2D(filters=32, kernel_size=(3, 3), input_shape=input_shape,␣
↪activation='leaky_relu')(layer5)
    layer7 = BatchNormalization()(layer6)
    layer8 = MaxPool2D(pool_size=(3, 3), padding='same')(layer7)
    layer9 = BatchNormalization()(layer8)
    layer10 = Flatten()(layer9)
    layer11 = Dense(128, activation='relu')(layer10)
    layer12 = BatchNormalization()(layer11)
    output = Dense(num_classes, activation='softmax')(layer12)
    layer13 = BatchNormalization()(output)
    model = Model(inputs = layer1, outputs = output)
    return model

input_shape = (96, 60, 3)
num_classes = 5
```

[29]:
```
model_02 = create_cnn_model(input_shape, num_classes)
```

```
model_02.compile(optimizer='adam', loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])
model_02.summary()
```

```
C:\Users\Mo\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kf
ra8p0\LocalCache\local-packages\Python311\site-
packages\keras\src\layers\convolutional\base_conv.py:99: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(
```

Model: "functional_9"

```
 Layer (type)                         Output Shape                          ␣
↪Param #

 input_layer_1 (InputLayer)           (None, 96, 60, 3)                       ␣
↪   0

 conv2d (Conv2D)                      (None, 94, 58, 32)                      ␣
↪896

 batch_normalization                  (None, 94, 58, 32)                      ␣
↪128
 (BatchNormalization)                                                         ␣
↪

 max_pooling2d (MaxPooling2D)         (None, 32, 20, 32)                      ␣
↪   0

 batch_normalization_1                (None, 32, 20, 32)                      ␣
↪128
 (BatchNormalization)                                                         ␣
↪

 conv2d_1 (Conv2D)                    (None, 30, 18, 32)                    ␣
↪9,248

 batch_normalization_2                (None, 30, 18, 32)                      ␣
↪128
 (BatchNormalization)                                                         ␣
↪

 max_pooling2d_1 (MaxPooling2D)       (None, 10, 6, 32)                       ␣
↪   0
```

```
 batch_normalization_3                  (None, 10, 6, 32)                       ␣
↪128
 (BatchNormalization)                                                           ␣
↪

 flatten_1 (Flatten)                    (None, 1920)                            ␣
↪  0

 dense_7 (Dense)                        (None, 128)                         ␣
↪245,888

 batch_normalization_4                  (None, 128)                             ␣
↪512
 (BatchNormalization)                                                           ␣
↪

 dense_8 (Dense)                        (None, 5)                               ␣
↪645



 Total params: 257,701 (1006.64 KB)


 Trainable params: 257,189 (1004.64 KB)


 Non-trainable params: 512 (2.00 KB)
```

I tried to use SparseCategoricalCrossentropy instade of categorical_crossentropy but becuase SparseCategoricalCrossentropy works on integers and the labels are in a one_hot representation i used categorical_crossentropy. I didn't find any other difference between them (they have the same math formula) so I decided to stick with categorical_crossentropy "Use this crossentropy loss function when there are two or more label classes. We expect labels to be provided in a one_hot representation. If you want to provide labels as integers, please use SparseCategoricalCrossentropy loss. There should be num_classes floating point values per feature, i.e., the shape of both y_pred and y_true are [batch_size, num_classes]." https://keras.io/api/losses/probabilistic_losses/

[30]:
```python
# Train CNN model
history_02 = model_02.fit(
    train_images_01, df_train_01['label'],
    epochs=50, batch_size=128,
    validation_data=(val_images_01, df_val_01['label']),
    callbacks=[early_stopping]
)
```

```
Epoch 1/50
4/4              2s 115ms/step -
```

```
accuracy: 0.2849 - loss: 2.0170 - val_accuracy: 0.0909 - val_loss: 1.6163
Epoch 2/50
4/4            0s 69ms/step -
accuracy: 0.6612 - loss: 0.8596 - val_accuracy: 0.0909 - val_loss: 1.6234
Epoch 3/50
4/4            0s 68ms/step -
accuracy: 0.7876 - loss: 0.6151 - val_accuracy: 0.2121 - val_loss: 1.6295
Epoch 4/50
4/4            0s 70ms/step -
accuracy: 0.8694 - loss: 0.4745 - val_accuracy: 0.2121 - val_loss: 1.6638
Epoch 5/50
4/4            0s 69ms/step -
accuracy: 0.8872 - loss: 0.3924 - val_accuracy: 0.0909 - val_loss: 1.6803
Epoch 6/50
4/4            0s 69ms/step -
accuracy: 0.9422 - loss: 0.2906 - val_accuracy: 0.0909 - val_loss: 1.7147
Epoch 7/50
4/4            0s 69ms/step -
accuracy: 0.9727 - loss: 0.2378 - val_accuracy: 0.0909 - val_loss: 1.7798
Epoch 8/50
4/4            0s 70ms/step -
accuracy: 0.9734 - loss: 0.1998 - val_accuracy: 0.0909 - val_loss: 1.8786
Epoch 9/50
4/4            0s 69ms/step -
accuracy: 0.9892 - loss: 0.1804 - val_accuracy: 0.0909 - val_loss: 2.0029
Epoch 10/50
4/4            0s 72ms/step -
accuracy: 0.9933 - loss: 0.1514 - val_accuracy: 0.2121 - val_loss: 2.1641
Epoch 11/50
4/4            0s 71ms/step -
accuracy: 0.9897 - loss: 0.1421 - val_accuracy: 0.2121 - val_loss: 2.3445
Epoch 12/50
4/4            0s 70ms/step -
accuracy: 0.9990 - loss: 0.1252 - val_accuracy: 0.2121 - val_loss: 2.2546
Epoch 13/50
4/4            0s 71ms/step -
accuracy: 0.9949 - loss: 0.1106 - val_accuracy: 0.2121 - val_loss: 2.1062
Epoch 14/50
4/4            0s 69ms/step -
accuracy: 0.9825 - loss: 0.1231 - val_accuracy: 0.2121 - val_loss: 1.7613
Epoch 15/50
4/4            0s 71ms/step -
accuracy: 0.9866 - loss: 0.1195 - val_accuracy: 0.2525 - val_loss: 1.7179
Epoch 16/50
4/4            0s 69ms/step -
accuracy: 0.9750 - loss: 0.1412 - val_accuracy: 0.2121 - val_loss: 1.7019
Epoch 17/50
4/4            0s 69ms/step -
```

```
accuracy: 0.9845 - loss: 0.1098 - val_accuracy: 0.2121 - val_loss: 1.8076
Epoch 18/50
4/4            0s 71ms/step -
accuracy: 0.9951 - loss: 0.0852 - val_accuracy: 0.2121 - val_loss: 2.2511
Epoch 19/50
4/4            0s 70ms/step -
accuracy: 1.0000 - loss: 0.0728 - val_accuracy: 0.2121 - val_loss: 2.6345
Epoch 20/50
4/4            0s 70ms/step -
accuracy: 1.0000 - loss: 0.0706 - val_accuracy: 0.2121 - val_loss: 2.8879
Epoch 21/50
4/4            0s 68ms/step -
accuracy: 0.9977 - loss: 0.0706 - val_accuracy: 0.2121 - val_loss: 2.9632
Epoch 22/50
4/4            0s 68ms/step -
accuracy: 0.9990 - loss: 0.0622 - val_accuracy: 0.2121 - val_loss: 2.8164
Epoch 23/50
4/4            0s 69ms/step -
accuracy: 1.0000 - loss: 0.0506 - val_accuracy: 0.2121 - val_loss: 2.6219
Epoch 24/50
4/4            0s 69ms/step -
accuracy: 1.0000 - loss: 0.0468 - val_accuracy: 0.2121 - val_loss: 2.3610
Epoch 25/50
4/4            0s 70ms/step -
accuracy: 1.0000 - loss: 0.0390 - val_accuracy: 0.2424 - val_loss: 2.2223
Epoch 26/50
4/4            0s 68ms/step -
accuracy: 0.9990 - loss: 0.0365 - val_accuracy: 0.2424 - val_loss: 2.2755
Epoch 26: early stopping
Restoring model weights from the end of the best epoch: 16.
```

0.5.3 RNN Model

[31]:
```
from tensorflow import keras
```

[32]:
```
batch_size = 256
input_dim = 60
units = 96
output_size = 10  # labels are from 0 to 9
```

[33]:
```
def create_rnn_model(input_shape, num_classes):
    model = keras.Sequential(
        [
            layers.Input(shape=input_shape),
            layers.Reshape((input_shape[0], input_shape[1] * input_shape[2])),
 ↪# Reshape input to (96, 60*3)
            layers.LSTM(96),
```

```
            layers.BatchNormalization(),
            layers.Dropout(rate=0.5),
            layers.Dense(num_classes, activation='softmax'),
        ]
    )

    return model

input_shape = (96, 60, 3)
num_classes = 5
```

[34]:
```
model_03 = create_rnn_model(input_shape, num_classes)
model_03.compile(optimizer='adamW', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
model_03.summary()
```

```
Model: "sequential_1"


 Layer (type)                           Output Shape                          ␣
 ↪Param #

 reshape (Reshape)                      (None, 96, 180)                         ␣
 ↪   0

 lstm (LSTM)                            (None, 96)                            ␣
 ↪106,368

 batch_normalization_6                  (None, 96)                              ␣
 ↪384
 (BatchNormalization)                                                           ␣
 ↪

 dropout (Dropout)                      (None, 96)                              ␣
 ↪   0

 dense_9 (Dense)                        (None, 5)                               ␣
 ↪485


 Total params: 107,237 (418.89 KB)


 Trainable params: 107,045 (418.14 KB)


 Non-trainable params: 192 (768.00 B)
```

[35]:
```
# Train RNN model
history_03 = model_03.fit(
    train_images_01, train_labels_01,
    epochs=50, batch_size=batch_size,
    validation_data=(val_images_01, val_labels_01),
    callbacks=[early_stopping]
)
```

```
Epoch 1/50
2/2              2s 414ms/step -
accuracy: 0.2309 - loss: 2.4859 - val_accuracy: 0.2525 - val_loss: 1.6106
Epoch 2/50
2/2              0s 69ms/step -
accuracy: 0.2737 - loss: 2.1071 - val_accuracy: 0.0909 - val_loss: 1.6124
Epoch 3/50
2/2              0s 72ms/step -
accuracy: 0.2573 - loss: 2.0642 - val_accuracy: 0.0909 - val_loss: 1.6149
Epoch 4/50
2/2              0s 72ms/step -
accuracy: 0.3067 - loss: 1.8908 - val_accuracy: 0.0909 - val_loss: 1.6177
Epoch 5/50
2/2              0s 71ms/step -
accuracy: 0.3012 - loss: 1.8974 - val_accuracy: 0.0909 - val_loss: 1.6205
Epoch 6/50
2/2              0s 71ms/step -
accuracy: 0.3308 - loss: 1.7737 - val_accuracy: 0.0909 - val_loss: 1.6233
Epoch 7/50
2/2              0s 71ms/step -
accuracy: 0.3548 - loss: 1.7541 - val_accuracy: 0.0909 - val_loss: 1.6262
Epoch 8/50
2/2              0s 71ms/step -
accuracy: 0.3264 - loss: 1.6885 - val_accuracy: 0.0909 - val_loss: 1.6295
Epoch 9/50
2/2              0s 69ms/step -
accuracy: 0.3268 - loss: 1.7516 - val_accuracy: 0.0909 - val_loss: 1.6330
Epoch 10/50
2/2              0s 74ms/step -
accuracy: 0.3111 - loss: 1.7024 - val_accuracy: 0.0909 - val_loss: 1.6363
Epoch 11/50
2/2              0s 73ms/step -
accuracy: 0.3611 - loss: 1.6435 - val_accuracy: 0.0909 - val_loss: 1.6397
Epoch 12/50
2/2              0s 70ms/step -
accuracy: 0.3563 - loss: 1.6674 - val_accuracy: 0.0909 - val_loss: 1.6438
Epoch 13/50
2/2              0s 71ms/step -
```

```
accuracy: 0.3858 - loss: 1.5547 - val_accuracy: 0.0909 - val_loss: 1.6478
Epoch 14/50
2/2              0s 69ms/step -
accuracy: 0.3652 - loss: 1.5855 - val_accuracy: 0.0909 - val_loss: 1.6519
Epoch 15/50
2/2              0s 71ms/step -
accuracy: 0.3887 - loss: 1.5776 - val_accuracy: 0.0909 - val_loss: 1.6559
Epoch 16/50
2/2              0s 69ms/step -
accuracy: 0.4064 - loss: 1.4803 - val_accuracy: 0.0909 - val_loss: 1.6599
Epoch 17/50
2/2              0s 74ms/step -
accuracy: 0.4182 - loss: 1.4697 - val_accuracy: 0.0909 - val_loss: 1.6643
Epoch 18/50
2/2              0s 71ms/step -
accuracy: 0.4157 - loss: 1.4436 - val_accuracy: 0.0909 - val_loss: 1.6686
Epoch 19/50
2/2              0s 71ms/step -
accuracy: 0.4341 - loss: 1.4370 - val_accuracy: 0.0909 - val_loss: 1.6725
Epoch 20/50
2/2              0s 71ms/step -
accuracy: 0.4689 - loss: 1.3592 - val_accuracy: 0.0909 - val_loss: 1.6766
Epoch 21/50
2/2              0s 73ms/step -
accuracy: 0.4740 - loss: 1.4511 - val_accuracy: 0.0909 - val_loss: 1.6812
Epoch 21: early stopping
Restoring model weights from the end of the best epoch: 11.
```

0.6 4. Evaluate the quality of multi-class classification by MLP, CNN and RNN neural networks on the test set using the quality indicator specified in the individual task, and output the neural network architecture with the best quality.

[36]:
```python
from sklearn.metrics import precision_score
```

[37]:
```python
test_loss_01, test_acc_01 = model_01.evaluate(test_images_01, test_labels_01)
test_loss_02, test_acc_02 = model_02.evaluate(test_images_01,␣
 ↪df_test_01['label'])
test_loss_03, test_acc_03 = model_03.evaluate(test_images_01, test_labels_01)
```

```
16/16              0s 1ms/step -
accuracy: 0.0432 - loss: 2.6087
16/16              0s 3ms/step -
accuracy: 0.4679 - loss: 1.5620
16/16              0s 6ms/step -
accuracy: 0.2575 - loss: 1.5528
```

[38]:
```
# Predictions for each model
pred_01 = model_01.predict(test_images_01)
pred_02 = model_02.predict(test_images_01)
pred_03 = model_03.predict(test_images_01)

# Convert predictions to class labels
pred_01_labels = np.argmax(pred_01, axis=1)
pred_02_labels = np.argmax(pred_02, axis=1)
pred_03_labels = np.argmax(pred_03, axis=1)
```

```
16/16            0s 4ms/step
16/16            0s 8ms/step
16/16            0s 13ms/step
```

[39]:
```
true_labels = df_test_01['label'].values
```

[40]:
```
precision_mlp = precision_score(true_labels, pred_01_labels, average='micro')
precision_cnn = precision_score(true_labels, pred_02_labels, average='micro')
precision_rnn = precision_score(true_labels, pred_03_labels, average='micro')
print(precision_mlp, precision_cnn, precision_rnn)
```

```
0.2012072434607646 0.289738430583501 0.2193158953722334
```

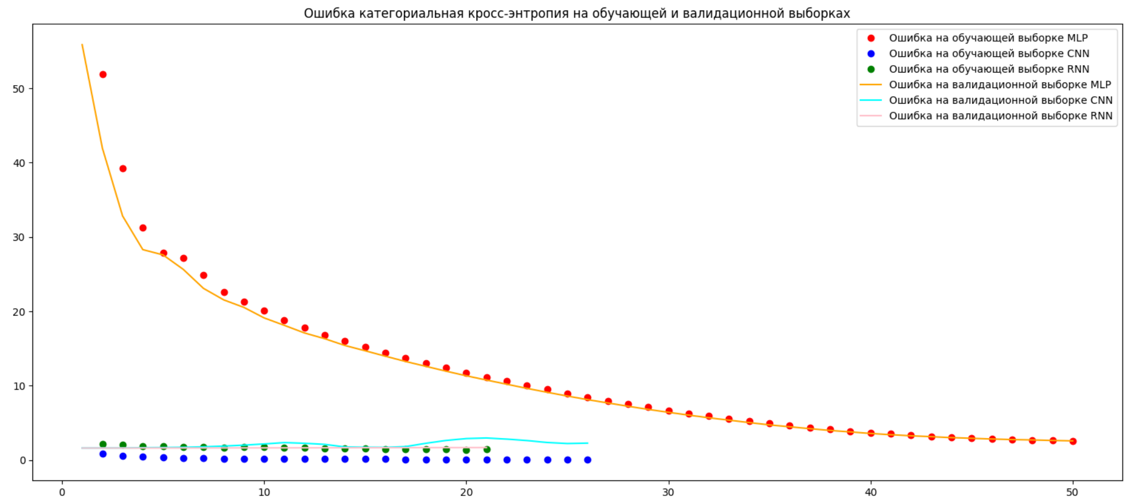
The RNN model showed the best results

0.7 5. Visualize the learning curves of the three built models for the loss rate on the validation set in one figure depending on the training epoch, labeling the axes and the figure and creating a legend. Use relative losses (losses divided by initial losses in the first epoch) for visualization.

[41]:
```
loss_01 = history_01.history["loss"]
loss_02 = history_02.history["loss"]
loss_03 = history_03.history["loss"]

val_loss_01 = history_01.history["val_loss"]
val_loss_02 = history_02.history["val_loss"]
val_loss_03 = history_03.history["val_loss"]

epochs_01 = range(1, len(loss_01) + 1)
epochs_02 = range(1, len(loss_02) + 1)
epochs_03 = range(1, len(loss_03) + 1)

plt.figure(figsize=(19, 8))
plt.plot(epochs_01[1:], loss_01[1:], "bo", color='red', label="
 ↪MLP")
plt.plot(epochs_02[1:], loss_02[1:], "bo", color='blue', label="
 ↪    CNN")
```

```
plt.plot(epochs_03[1:], loss_03[1:], "bo", color='green', label="
     RNN")

plt.plot(epochs_01, val_loss_01, "b", color='orange', label="
 MLP")
plt.plot(epochs_02, val_loss_02, "b", color='cyan', label="
 CNN")
plt.plot(epochs_03, val_loss_03, "b", color='pink', label="
 RNN")

plt.title("              -                            ")
plt.legend()
plt.show()
```

```
C:\Users\Mo\AppData\Local\Temp\ipykernel_11476\2298408522.py:14: UserWarning:
color is redundantly defined by the 'color' keyword argument and the fmt string
"bo" (-> color='b'). The keyword argument will take precedence.
  plt.plot(epochs_01[1:], loss_01[1:], "bo", color='red', label="
         MLP")
C:\Users\Mo\AppData\Local\Temp\ipykernel_11476\2298408522.py:15: UserWarning:
color is redundantly defined by the 'color' keyword argument and the fmt string
"bo" (-> color='b'). The keyword argument will take precedence.
  plt.plot(epochs_02[1:], loss_02[1:], "bo", color='blue', label="
         CNN")
C:\Users\Mo\AppData\Local\Temp\ipykernel_11476\2298408522.py:16: UserWarning:
color is redundantly defined by the 'color' keyword argument and the fmt string
"bo" (-> color='b'). The keyword argument will take precedence.
  plt.plot(epochs_03[1:], loss_03[1:], "bo", color='green', label="
         RNN")
C:\Users\Mo\AppData\Local\Temp\ipykernel_11476\2298408522.py:18: UserWarning:
color is redundantly defined by the 'color' keyword argument and the fmt string
"b" (-> color=(0.0, 0.0, 1.0, 1)). The keyword argument will take precedence.
  plt.plot(epochs_01, val_loss_01, "b", color='orange', label="
           MLP")
C:\Users\Mo\AppData\Local\Temp\ipykernel_11476\2298408522.py:19: UserWarning:
color is redundantly defined by the 'color' keyword argument and the fmt string
"b" (-> color=(0.0, 0.0, 1.0, 1)). The keyword argument will take precedence.
  plt.plot(epochs_02, val_loss_02, "b", color='cyan', label="
           CNN")
C:\Users\Mo\AppData\Local\Temp\ipykernel_11476\2298408522.py:20: UserWarning:
color is redundantly defined by the 'color' keyword argument and the fmt string
"b" (-> color=(0.0, 0.0, 1.0, 1)). The keyword argument will take precedence.
  plt.plot(epochs_03, val_loss_03, "b", color='pink', label="
           RNN")
```

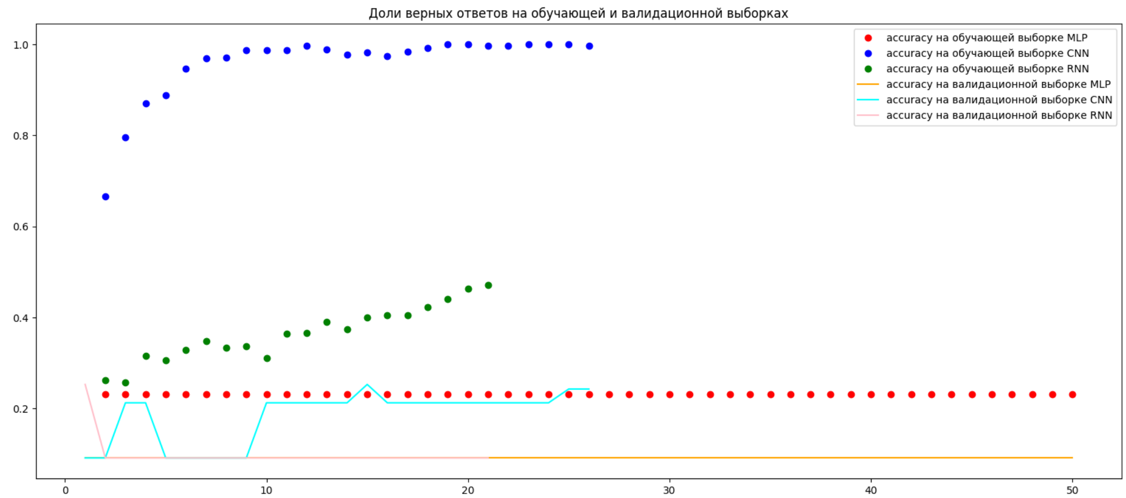


0.8 6. Visualize the learning curves of the three constructed models for the percentage of correct answers on the validation set in one figure depending on the training epoch, labeling the axes and the figure and creating a legend.

[42]:
```
acc_01 = history_01.history["accuracy"]
acc_02 = history_02.history["accuracy"]
acc_03 = history_03.history["accuracy"]

val_acc_01 = history_01.history["val_accuracy"]
val_acc_02 = history_02.history["val_accuracy"]
val_acc_03 = history_03.history["val_accuracy"]

epochs_01 = range(1, len(acc_01) + 1)
epochs_02 = range(1, len(acc_02) + 1)
epochs_03 = range(1, len(acc_03) + 1)

plt.figure(figsize=(19, 8))
plt.plot(epochs_01[1:], acc_01[1:], "bo", color='red', label="accuracy ␣
↪    MLP")
plt.plot(epochs_02[1:], acc_02[1:], "bo", color='blue', label="accuracy ␣
↪    CNN")
plt.plot(epochs_03[1:], acc_03[1:], "bo", color='green', label="accuracy ␣
↪          RNN")

plt.plot(epochs_01, val_acc_01, "b", color='orange', label="accuracy ␣
↪    MLP")
plt.plot(epochs_02, val_acc_02, "b", color='cyan', label="accuracy ␣
↪    CNN")
```

```
plt.plot(epochs_03, val_acc_03, "b", color='pink', label="accuracy ␣
↪     RNN")

plt.title("                                  ")
plt.legend()
plt.show()
```

```
C:\Users\Mo\AppData\Local\Temp\ipykernel_11476\4070226802.py:14: UserWarning:
color is redundantly defined by the 'color' keyword argument and the fmt string
"bo" (-> color='b'). The keyword argument will take precedence.
  plt.plot(epochs_01[1:], acc_01[1:], "bo", color='red', label="accuracy
         MLP")
C:\Users\Mo\AppData\Local\Temp\ipykernel_11476\4070226802.py:15: UserWarning:
color is redundantly defined by the 'color' keyword argument and the fmt string
"bo" (-> color='b'). The keyword argument will take precedence.
  plt.plot(epochs_02[1:], acc_02[1:], "bo", color='blue', label="accuracy
         CNN")
C:\Users\Mo\AppData\Local\Temp\ipykernel_11476\4070226802.py:16: UserWarning:
color is redundantly defined by the 'color' keyword argument and the fmt string
"bo" (-> color='b'). The keyword argument will take precedence.
  plt.plot(epochs_03[1:], acc_03[1:], "bo", color='green', label="accuracy
         RNN")
C:\Users\Mo\AppData\Local\Temp\ipykernel_11476\4070226802.py:18: UserWarning:
color is redundantly defined by the 'color' keyword argument and the fmt string
"b" (-> color=(0.0, 0.0, 1.0, 1)). The keyword argument will take precedence.
  plt.plot(epochs_01, val_acc_01, "b", color='orange', label="accuracy
           MLP")
C:\Users\Mo\AppData\Local\Temp\ipykernel_11476\4070226802.py:19: UserWarning:
color is redundantly defined by the 'color' keyword argument and the fmt string
"b" (-> color=(0.0, 0.0, 1.0, 1)). The keyword argument will take precedence.
  plt.plot(epochs_02, val_acc_02, "b", color='cyan', label="accuracy
           CNN")
C:\Users\Mo\AppData\Local\Temp\ipykernel_11476\4070226802.py:20: UserWarning:
color is redundantly defined by the 'color' keyword argument and the fmt string
"b" (-> color=(0.0, 0.0, 1.0, 1)). The keyword argument will take precedence.
  plt.plot(epochs_03, val_acc_03, "b", color='pink', label="accuracy
           RNN")
```


Доли верных ответов на обучающей и валидационной выборках
accuracy на обучающей выборке MLP
accuracy на обучающей выборке CNN
accuracy на обучающей выборке RNN
accuracy на валидационной выборке MLP
accuracy на валидационной выборке CNN
accuracy на валидационной выборке RNN
1.0
0.8
0.6
0.4
0.2
0
10
20
30
40
50