Summer 2024: CS5720

Neural Networks & Deep Learning

ICP-5

Name: Shiva Chaitanya Reddy Gudipati

#700s: 700756331

GitHub Link: https://github.com/Shiva-labs/ICP5

Programming elements:

1. Basics of Autoencoders
2. Role of Autoencoders in unsupervised learning
3. Types of Autoencoders
4. Use case: Simple autoencoder-Reconstructing the existing image, which will contain most important features of the image
5. Use case: Stacked autoencoder

```
from keras.layers import Input, Dense
from keras.models import Model
from keras.regularizers import l2
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping

# Enhanced Autoencoder Model
input_img = Input(shape=(784,))
encoded = Dense(128, activation='LeakyReLU',
kernel_regularizer=l2(0.001))(input_img)  # Deeper, L2 regularization
encoded = Dense(64, activation='LeakyReLU',
kernel_regularizer=l2(0.001))(encoded)
encoded = Dense(encoding_dim, activation='LeakyReLU')(encoded)   #
Bottleneck layer

decoded = Dense(64, activation='LeakyReLU')(encoded)
decoded = Dense(128, activation='LeakyReLU')(decoded)
decoded = Dense(784, activation='sigmoid')(decoded)
```

```python
autoencoder = Model(input_img, decoded)
optimizer = Adam(learning_rate=0.001)  # Start with a higher learning rate
autoencoder.compile(optimizer=optimizer, loss='binary_crossentropy')

# Early Stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)

autoencoder.fit(x_train, x_train,
                epochs=50,  # Increase epochs for potential better results
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test),
                callbacks=[early_stopping])
```

Output:

```
Epoch 1/50
235/235 [==============================] - 4s 6ms/step - loss: 0.4496 - val_loss: 0.3517
Epoch 2/50
235/235 [==============================] - 1s 5ms/step - loss: 0.3324 - val_loss: 0.3241
Epoch 3/50
235/235 [==============================] - 1s 5ms/step - loss: 0.3185 - val_loss: 0.3170
Epoch 4/50
235/235 [==============================] - 1s 5ms/step - loss: 0.3126 - val_loss: 0.3126
Epoch 5/50
235/235 [==============================] - 1s 5ms/step - loss: 0.3093 - val_loss: 0.3099
Epoch 6/50
235/235 [==============================] - 1s 5ms/step - loss: 0.3070 - val_loss: 0.3080
Epoch 7/50
235/235 [==============================] - 1s 5ms/step - loss: 0.3052 - val_loss: 0.3067
Epoch 8/50
235/235 [==============================] - 2s 7ms/step - loss: 0.3037 - val_loss: 0.3056
Epoch 9/50
235/235 [==============================] - 2s 7ms/step - loss: 0.3025 - val_loss: 0.3048
Epoch 10/50
235/235 [==============================] - 1s 5ms/step - loss: 0.3014 - val_loss: 0.3025
Epoch 11/50
235/235 [==============================] - 1s 5ms/step - loss: 0.3002 - val_loss: 0.3026
Epoch 12/50
235/235 [==============================] - 1s 5ms/step - loss: 0.2995 - val_loss: 0.3008
Epoch 13/50
235/235 [==============================] - 1s 5ms/step - loss: 0.2988 - val_loss: 0.3002
Epoch 14/50
235/235 [==============================] - 1s 5ms/step - loss: 0.2980 - val_loss: 0.2994
Epoch 15/50
235/235 [==============================] - 1s 5ms/step - loss: 0.2974 - val_loss: 0.2997
Epoch 16/50
235/235 [==============================] - 1s 5ms/step - loss: 0.2967 - val_loss: 0.2986
Epoch 17/50
235/235 [==============================] - 1s 5ms/step - loss: 0.2961 - val_loss: 0.2984
Epoch 18/50
235/235 [==============================] - 2s 6ms/step - loss: 0.2956 - val_loss: 0.2973
```

```
Epoch 19/50
235/235 [==============================] - 2s 7ms/step - loss: 0.2958 - val_loss: 0.2971
Epoch 20/50
235/235 [==============================] - 1s 5ms/step - loss: 0.2947 - val_loss: 0.2968
Epoch 21/50
235/235 [==============================] - 1s 5ms/step - loss: 0.2943 - val_loss: 0.2964
Epoch 22/50
235/235 [==============================] - 1s 5ms/step - loss: 0.2941 - val_loss: 0.2964
Epoch 23/50
235/235 [==============================] - 1s 5ms/step - loss: 0.2938 - val_loss: 0.2958
Epoch 24/50
235/235 [==============================] - 1s 5ms/step - loss: 0.2936 - val_loss: 0.2953
Epoch 25/50
235/235 [==============================] - 1s 5ms/step - loss: 0.2932 - val_loss: 0.2954
Epoch 26/50
235/235 [==============================] - 1s 5ms/step - loss: 0.2929 - val_loss: 0.2955
Epoch 27/50
235/235 [==============================] - 1s 5ms/step - loss: 0.3042 - val_loss: 0.2968
Epoch 28/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2937 - val_loss: 0.2951
Epoch 29/50
235/235 [==============================] - 2s 7ms/step - loss: 0.2926 - val_loss: 0.2949
Epoch 30/50
235/235 [==============================] - 2s 7ms/step - loss: 0.2923 - val_loss: 0.2947
Epoch 31/50
235/235 [==============================] - 2s 8ms/step - loss: 0.2919 - val_loss: 0.2942
Epoch 32/50
235/235 [==============================] - 2s 8ms/step - loss: 0.2917 - val_loss: 0.2941
Epoch 33/50
235/235 [==============================] - 1s 5ms/step - loss: 0.2916 - val_loss: 0.2940
Epoch 34/50
235/235 [==============================] - 1s 5ms/step - loss: 0.2912 - val_loss: 0.2935
Epoch 35/50
235/235 [==============================] - 1s 5ms/step - loss: 0.2908 - val_loss: 0.2927
Epoch 36/50
235/235 [==============================] - 1s 5ms/step - loss: 0.2905 - val_loss: 0.2924
Epoch 37/50
235/235 [==============================] - 2s 6ms/step - loss: 0.2901 - val_loss: 0.2922
Epoch 38/50
235/235 [==============================] - 2s 9ms/step - loss: 0.2900 - val_loss: 0.2921
Epoch 39/50
235/235 [==============================] - 2s 9ms/step - loss: 0.2896 - val_loss: 0.2914
Epoch 40/50
235/235 [==============================] - 2s 9ms/step - loss: 0.2895 - val_loss: 0.2916
Epoch 41/50
235/235 [==============================] - 2s 9ms/step - loss: 0.2890 - val_loss: 0.2910
Epoch 42/50
235/235 [==============================] - 2s 9ms/step - loss: 0.2888 - val_loss: 0.2910
Epoch 43/50
235/235 [==============================] - 1s 5ms/step - loss: 0.2886 - val_loss: 0.2902
Epoch 44/50
235/235 [==============================] - 2s 7ms/step - loss: 0.2885 - val_loss: 0.2906
Epoch 45/50
235/235 [==============================] - 2s 7ms/step - loss: 0.2883 - val_loss: 0.2902
Epoch 46/50
```

235/235 [==============================] - 1s 5ms/step - loss: 0.2880 - val_loss: 0.2897
Epoch 47/50
235/235 [==============================] - 1s 5ms/step - loss: 0.2881 - val_loss: 0.2908
Epoch 48/50
235/235 [==============================] - 1s 5ms/step - loss: 0.2876 - val_loss: 0.2895
Epoch 49/50
235/235 [==============================] - 1s 5ms/step - loss: 0.2876 - val_loss: 0.2894
Epoch 50/50
235/235 [==============================] - 1s 5ms/step - loss: 0.3174 - val_loss: 0.2990
<keras.src.callbacks.History at 0x7d61bcfcead0>

Add one more hidden layer to autoencoder

```
from keras.layers import Input, Dense
from keras.models import Model
from keras.datasets import mnist, fashion_mnist
import numpy as np

# this is the size of our encoded representations
encoding_dim = 32
# this is our input placeholder
input_img = Input(shape=(784,))
    # "encoded" is the encoded representation of the input
    encoded = Dense(encoding_dim, activation='relu')(input_img)

    # Adding an additional hidden layer
    hidden_layer_dim = 64
    hidden_layer = Dense(hidden_layer_dim, activation='relu')(encoded)

    # "decoded" is the lossy reconstruction of the input, now connected to the hidden layer instead of 'encoded'
    decoded = Dense(784, activation='sigmoid')(hidden_layer)

    # this model maps an input to its reconstruction
    autoencoder = Model(input_img, decoded)

    # this model maps an input to its encoded representation
    autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy')
    # Load and prepare the data
    (x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
    x_train = x_train.astype('float32') / 255.
    x_test = x_test.astype('float32') / 255.
```

```
    x_train = x_train.reshape((len(x_train),
np.prod(x_train.shape[1:])))
    x_test = x_test.reshape((len(x_test),
np.prod(x_test.shape[1:])))

    # Train the model
    autoencoder.fit(x_train, x_train,
                    epochs=5,
                    batch_size=256,
                    shuffle=True,
                    validation_data=(x_test, x_test))
```

Output:

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 [==============================] - 1s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 [==============================] - 0s 0us/step
Epoch 1/5
235/235 [==============================] - 12s 22ms/step - loss: 0.6948 - val_loss: 0.6947
Epoch 2/5
235/235 [==============================] - 3s 11ms/step - loss: 0.6946 - val_loss: 0.6946
Epoch 3/5
235/235 [==============================] - 2s 10ms/step - loss: 0.6945 - val_loss: 0.6944
Epoch 4/5
235/235 [==============================] - 2s 8ms/step - loss: 0.6944 - val_loss: 0.6943
Epoch 5/5
235/235 [==============================] - 1s 5ms/step - loss: 0.6942 - val_loss: 0.6941
<keras.src.callbacks.History at 0x7bf9c9d2df90>

Do the prediction on the test data and then visualize one of the reconstructed version of that test data. Also, visualize the same test data before reconstruction using Matplotlib

```
from keras.layers import Input, Dense, Dropout
from keras.models import Model
from keras.callbacks import EarlyStopping, ReduceLROnPlateau
from keras.datasets import fashion_mnist
import numpy as np
```

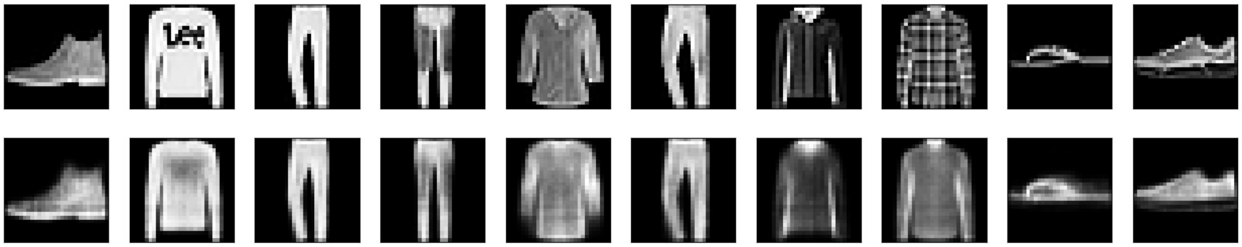
```
import matplotlib.pyplot as plt

# ... (Data loading and preparation remains the same)

# Enhanced model architecture
input_img = Input(shape=(784,))
encoded = Dense(128, activation='relu')(input_img)  # More units
encoded = Dropout(0.2)(encoded)                      # Dropout for
regularization
encoded = Dense(64, activation='relu')(encoded)
hidden_layer = Dense(128, activation='relu')(encoded)  # Deeper
architecture
decoded = Dense(784, activation='sigmoid')(hidden_layer)

autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

# Callbacks for improved training
early_stopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.1,
patience=3)

# Training with callbacks
autoencoder.fit(x_train, x_train,
                epochs=15,                  # Increased epochs for deeper
model
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test),
                callbacks=[early_stopping, lr_scheduler])

# Predict on the test data
decoded_imgs = autoencoder.predict(x_test)

# Visualize the original and reconstructed data
n = 10  # how many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
    # display original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
```

```
    # display reconstruction
    ax = plt.subplot(2, n, i + n + 1)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```

Output:

```
Epoch 1/15
235/235 [==============================] - 7s 13ms/step - loss: 0.3797 -
val_loss: 0.3174 - lr: 0.0010
Epoch 2/15
235/235 [==============================] - 2s 9ms/step - loss: 0.3135 -
val_loss: 0.3037 - lr: 0.0010
Epoch 3/15
235/235 [==============================] - 2s 9ms/step - loss: 0.3049 -
val_loss: 0.2991 - lr: 0.0010
Epoch 4/15
235/235 [==============================] - 2s 9ms/step - loss: 0.3003 -
val_loss: 0.2962 - lr: 0.0010
Epoch 5/15
235/235 [==============================] - 2s 8ms/step - loss: 0.2968 -
val_loss: 0.2947 - lr: 0.0010
Epoch 6/15
235/235 [==============================] - 2s 8ms/step - loss: 0.2942 -
val_loss: 0.2957 - lr: 0.0010
Epoch 7/15
235/235 [==============================] - 3s 11ms/step - loss: 0.2924 -
val_loss: 0.2934 - lr: 0.0010
Epoch 8/15
235/235 [==============================] - 2s 10ms/step - loss: 0.2911 -
val_loss: 0.2924 - lr: 0.0010
Epoch 9/15
235/235 [==============================] - 2s 9ms/step - loss: 0.2900 -
val_loss: 0.2906 - lr: 0.0010
Epoch 10/15
235/235 [==============================] - 1s 6ms/step - loss: 0.2891 -
val_loss: 0.2894 - lr: 0.0010
Epoch 11/15
235/235 [==============================] - 2s 7ms/step - loss: 0.2884 -
val_loss: 0.2903 - lr: 0.0010
Epoch 12/15
235/235 [==============================] - 2s 7ms/step - loss: 0.2877 -
val_loss: 0.2889 - lr: 0.0010
Epoch 13/15
235/235 [==============================] - 2s 7ms/step - loss: 0.2872 -
val_loss: 0.2920 - lr: 0.0010
Epoch 14/15
235/235 [==============================] - 2s 7ms/step - loss: 0.2867 -
val_loss: 0.2893 - lr: 0.0010
Epoch 15/15
```

```
235/235 [==============================] - 2s 10ms/step - loss: 0.2862 -
val_loss: 0.2865 - lr: 0.0010
313/313 [==============================] - 1s 2ms/step
```



Repeat the question 2 on the denoisening autoencoder

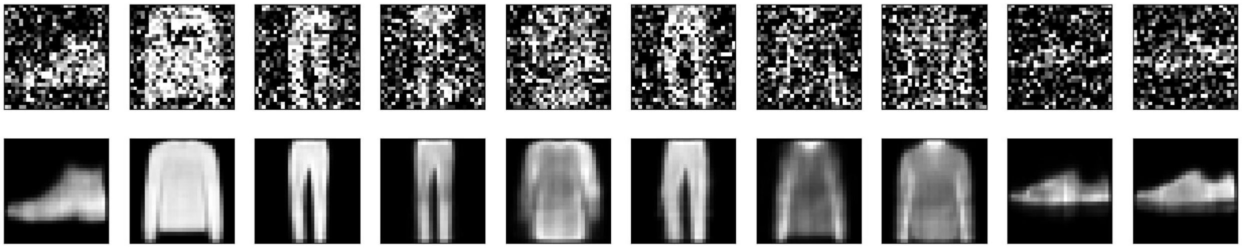
```
from keras.layers import Input, Dense, Dropout
from keras.models import Model
from keras.callbacks import EarlyStopping, ReduceLROnPlateau
from keras.datasets import fashion_mnist
import numpy as np
import matplotlib.pyplot as plt


# Model architecture with regularization
encoding_dim = 64  # Increased encoding dimension for better
representation
input_img = Input(shape=(784,))
encoded = Dense(encoding_dim, activation='relu')(input_img)
encoded = Dropout(0.2)(encoded)  # Add dropout for regularization
decoded = Dense(784, activation='sigmoid')(encoded)

autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

# Callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.1,
patience=3)

# Noise introduction
noise_factor = 0.5 # You can adjust this for more/less noise
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0,
scale=1.0, size=x_train.shape)
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0,
size=x_test.shape)
x_train_noisy = np.clip(x_train_noisy, 0., 1.)
```

```python
x_test_noisy = np.clip(x_test_noisy, 0., 1.)

# Train the model
autoencoder.fit(x_train_noisy, x_train,  # Train on noisy input, target is
clean
                epochs=20,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test_noisy, x_test),
                callbacks=[early_stopping, lr_scheduler])

# Predict on the noisy test data
decoded_imgs = autoencoder.predict(x_test_noisy)

# Visualize the noisy input and the reconstructed data
n = 10  # How many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
    # Display noisy input
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test_noisy[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # Display reconstruction
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```

Output:

```
Epoch 1/20
235/235 [==============================] - 3s 5ms/step - loss: 0.4210 -
val_loss: 0.3453 - lr: 0.0010
Epoch 2/20
235/235 [==============================] - 1s 4ms/step - loss: 0.3433 -
val_loss: 0.3300 - lr: 0.0010
Epoch 3/20
235/235 [==============================] - 1s 5ms/step - loss: 0.3331 -
val_loss: 0.3238 - lr: 0.0010
Epoch 4/20
235/235 [==============================] - 1s 5ms/step - loss: 0.3275 -
val_loss: 0.3195 - lr: 0.0010
```

```
Epoch 5/20
235/235 [==============================] - 1s 5ms/step - loss: 0.3241 -
val_loss: 0.3163 - lr: 0.0010
Epoch 6/20
235/235 [==============================] - 1s 5ms/step - loss: 0.3217 -
val_loss: 0.3141 - lr: 0.0010
Epoch 7/20
235/235 [==============================] - 1s 5ms/step - loss: 0.3197 -
val_loss: 0.3122 - lr: 0.0010
Epoch 8/20
235/235 [==============================] - 1s 5ms/step - loss: 0.3184 -
val_loss: 0.3110 - lr: 0.0010
Epoch 9/20
235/235 [==============================] - 2s 7ms/step - loss: 0.3171 -
val_loss: 0.3098 - lr: 0.0010
Epoch 10/20
235/235 [==============================] - 2s 7ms/step - loss: 0.3161 -
val_loss: 0.3088 - lr: 0.0010
Epoch 11/20
235/235 [==============================] - 1s 5ms/step - loss: 0.3153 -
val_loss: 0.3079 - lr: 0.0010
Epoch 12/20
235/235 [==============================] - 1s 5ms/step - loss: 0.3146 -
val_loss: 0.3074 - lr: 0.0010
Epoch 13/20
235/235 [==============================] - 1s 5ms/step - loss: 0.3139 -
val_loss: 0.3063 - lr: 0.0010
Epoch 14/20
235/235 [==============================] - 1s 5ms/step - loss: 0.3134 -
val_loss: 0.3061 - lr: 0.0010
Epoch 15/20
235/235 [==============================] - 1s 5ms/step - loss: 0.3129 -
val_loss: 0.3052 - lr: 0.0010
Epoch 16/20
235/235 [==============================] - 1s 5ms/step - loss: 0.3125 -
val_loss: 0.3047 - lr: 0.0010
Epoch 17/20
235/235 [==============================] - 1s 5ms/step - loss: 0.3119 -
val_loss: 0.3046 - lr: 0.0010
Epoch 18/20
235/235 [==============================] - 1s 5ms/step - loss: 0.3116 -
val_loss: 0.3040 - lr: 0.0010
Epoch 19/20
235/235 [==============================] - 1s 5ms/step - loss: 0.3111 -
val_loss: 0.3038 - lr: 0.0010
Epoch 20/20
235/235 [==============================] - 2s 7ms/step - loss: 0.3109 -
val_loss: 0.3034 - lr: 0.0010
313/313 [==============================] - 1s 3ms/step
```



plot loss and accuracy using the history object

```
from keras.layers import Input, Dense
from keras.models import Model
from keras.datasets import fashion_mnist
from keras.utils import to_categorical
import numpy as np
import matplotlib.pyplot as plt
from keras.optimizers import Adam

# Load and prepare the Fashion MNIST data
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
x_train = x_train.reshape(-1, 784).astype('float32') / 255
x_test = x_test.reshape(-1, 784).astype('float32') / 255

# Convert labels to one-hot encoding
num_classes = 10
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)
# Model architecture
input_img = Input(shape=(784,))
encoded = Dense(128, activation='relu')(input_img)
decoded = Dense(10, activation='softmax')(encoded)  # Classification layer

model = Model(input_img, decoded)
model.compile(optimizer=Adam(learning_rate=0.001),
loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=256,
                    shuffle=True,
                    validation_data=(x_test, y_test))
```

```python
# Plotting the training and validation loss
plt.figure(figsize=(10, 5))

# Plotting training and validation accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Plotting training and validation loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.tight_layout()
plt.show()
```

Output:

```
Epoch 1/10
235/235 [==============================] - 2s 6ms/step - loss: 0.6263 -
accuracy: 0.7865 - val_loss: 0.4995 - val_accuracy: 0.8282
Epoch 2/10
235/235 [==============================] - 2s 7ms/step - loss: 0.4339 -
accuracy: 0.8517 - val_loss: 0.4470 - val_accuracy: 0.8445
Epoch 3/10
235/235 [==============================] - 1s 4ms/step - loss: 0.3932 -
accuracy: 0.8637 - val_loss: 0.4276 - val_accuracy: 0.8493
Epoch 4/10
235/235 [==============================] - 1s 4ms/step - loss: 0.3675 -
accuracy: 0.8704 - val_loss: 0.3909 - val_accuracy: 0.8621
Epoch 5/10
235/235 [==============================] - 1s 4ms/step - loss: 0.3457 -
accuracy: 0.8774 - val_loss: 0.3841 - val_accuracy: 0.8650
Epoch 6/10
235/235 [==============================] - 1s 4ms/step - loss: 0.3290 -
accuracy: 0.8833 - val_loss: 0.3816 - val_accuracy: 0.8607
Epoch 7/10
235/235 [==============================] - 1s 6ms/step - loss: 0.3173 -
accuracy: 0.8873 - val_loss: 0.3644 - val_accuracy: 0.8716
Epoch 8/10
235/235 [==============================] - 1s 6ms/step - loss: 0.3048 -
accuracy: 0.8910 - val_loss: 0.3528 - val_accuracy: 0.8756
Epoch 9/10
```

```
235/235 [==============================] - 1s 5ms/step - loss: 0.2947 -
accuracy: 0.8938 - val_loss: 0.3535 - val_accuracy: 0.8749
Epoch 10/10
235/235 [==============================] - 1s 4ms/step - loss: 0.2876 -
accuracy: 0.8969 - val_loss: 0.3580 - val_accuracy: 0.8724
```