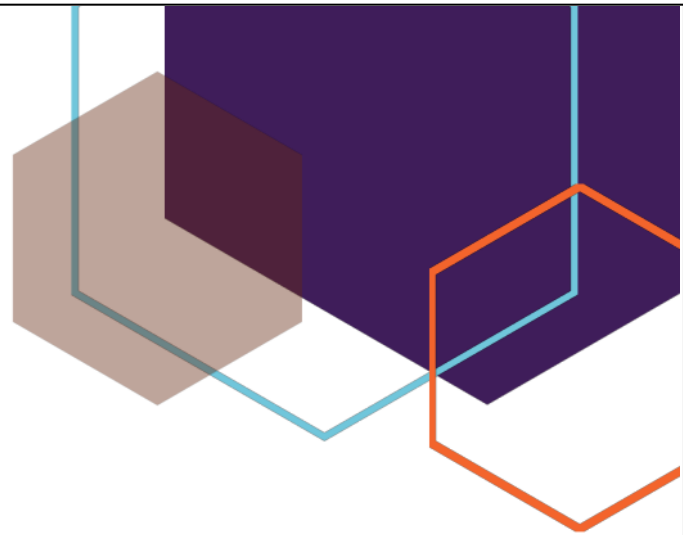


Machine Learning 2

by William Henry Baker

FILE	3611_WILLIAM_HENRY_BAKER_MACHINE_LEARNING_2_316598_397008058.DOCX (838.37K)		
TIME SUBMITTED	18-JAN-2021 11:10PM (UTC+0000)	WORD COUNT	4083
SUBMISSION ID	142623384	CHARACTER COUNT	27735



Machine Learning 2



Application of Machine Learning (ACCA7025)

William Baker – 1703131

Software Engineering and Artificial Intelligence

Gordon Dickers

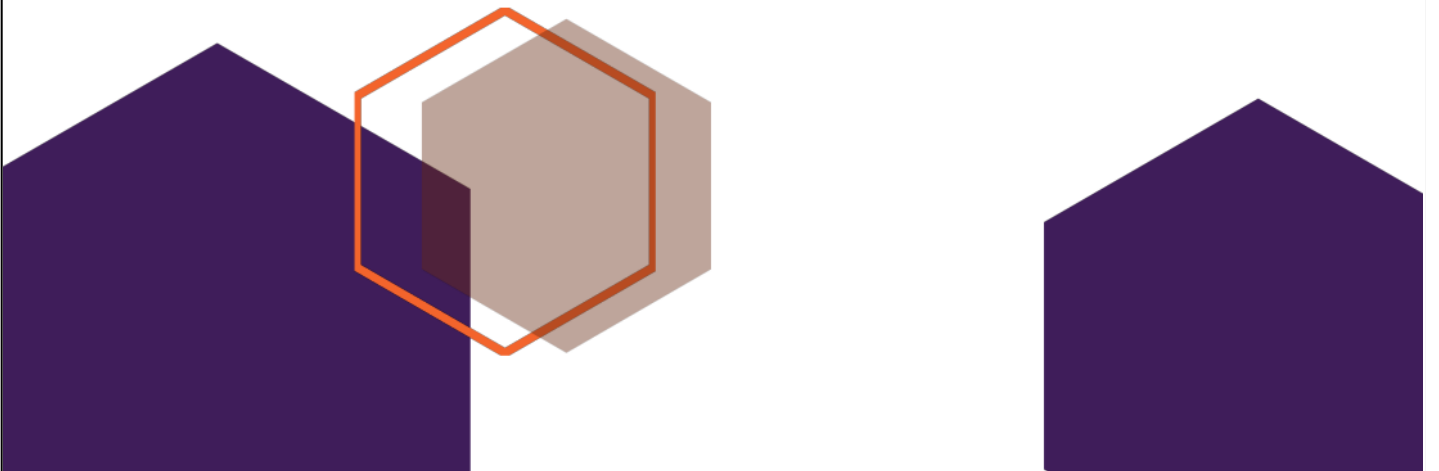


Table of Contents

Table of Figures	2
Abstract	3
1. Introduction	3
2. Literature Review	3
2.1. Deep Neural Networks.....	3
2.2. Convolutional Neural Networks.....	4
2.2.1. Convolutional Layer	4
2.3. Hyper-parameters.....	5
2.3.1. Pooling Layer	5
2.3.2. Learning Rate	5
2.3.3. Dropout.....	6
2.3.4. Activation function	6
3. Experiments	6
3.1. Examining the Data.....	6
3.2. Performance Measure	7
3.3. Testing Model Architecture	7
3.3.1. Starting Architecture	7
3.3.2. Increasing the Learning Rate.....	8
3.3.3. Increasing the Hidden Layers.....	8
3.3.4. Increasing the number of Convolutional Layers	8
3.3.5. Convolutional Strides and Early Stopping.....	9
3.3.6. Decreased Dropout.....	10
3.3.7. Final Model Architecture	10
3.4. Evaluating the Model.....	11
4. Conclusion.....	11
Bibliography	12
Appendices	13
A. Model Training Code	13

Table of Figures

Figure 1 - The Convolutional Layer Kernel moving across an image (Source: https://github.com/vdumoulin/conv_arithmetic)	4
Figure 2 - An example of how max-pooling affects the layer.....	5
Figure 3 - The layer setup for a basic CNN.....	7
Figure 4 - A Tensor Board chart showing how learning rate affects the model accuracy	8
Figure 5 - Training and Validation accuracy plotted for a larger Neural Network.....	9
Figure 6 - The Training and Validation accuracy after decreasing the dropout	10
Figure 7 - The Training and Validation accuracy of the finalized neural network architecture..	10
Figure 8 - The Final Model accuracy tested using a separate test set	11

Abstract

Neural Networks are intertwined in a considerable number of software products we use on a daily basis. Deep Learning (DL) is widely used for a number of different applications, such as computer vision and language processing. Deep Learning affects many of our lives with its use in medical devices, automated driving, personal assistants, and more.

1. Introduction

Neural Networks can be used for many tasks, from gaming bots to image recognition to self-driving vehicles. This report aims to look into how neural networks can be efficiently used for image classification and what types of neural networks are the best for this task. It will dive into the hyperparameters for neural networks that can be used to tune the models to get the best results. Experiments will be carried out using Convolutional Neural Networks to detect facial expressions in given images, hyper-parameter tuning and hidden layer selection will be reviewed and experimented with to come to the best result.

2. Literature Review

There are many different types of Artificial Neural Networks (ANNs) used in modern computing, and they all have their uses in specific fields. This literature review intends to look into Deep Neural Networks (DNNs) and Convolutional Neural Networks (CNNs) to see how they differ and how they can be used to classify a dataset of images successfully.

2.1. Deep Neural Networks

Deep Neural Networks (DNNs) are a type of Artificial Neural Network (ANN) that use multiple layers between the input and output. Deep Neural Networks are more complex and computationally demanding than other types of neural networks; this is due to their multi-layer architecture. There are often more neurons and a lot more parameters than are found in other

neural networks. Generally, DNNs are feed-forward networks, meaning the data flows forward from the input layer through to the output layer without looping back. [1]

2.2. Convolutional Neural Networks

Convolutional Neural Networks (CNNs), otherwise known as ConvNets, are a class of DNN, they are primarily used for pattern recognition in images. CNN's are inspired by biology; the connectivity between the neurons resembles what is found in the visual cortex of animals. [Ref] In a CNN, the input layer is comprised of the pixels of the input image. This could be $48 \times 48 \times 1$ (2304) neurons for a greyscale image since greyscale only has a single depth channel; for RGB, this would be $48 \times 48 \times 3$ (6912) neurons, it is easy to see how you could get a massive number of neurons if the image resolution were bigger. CNN's use partially connected layers to solve this; they reduce the image into a form that is easier to process by applying filters. [2]

2.2.1. Convolutional Layer

A convolutional layer is the first layer of a CNN; it is used to perform the convolution operation. The convolutional layer "focuses" in on a certain area of the input image, using a filter (kernel) which is moves across the image. Kernel size is used to define the area of focus. Different kernel sizes can be used to get different outputs. A kernel filter size of 3×3 is the most common option as it provides a good result and runs fast.

In the past, kernel sizes 5×5 and even 11×11 have been used, but it can increase training times to weeks, which is much of the reason it has been phased out in recent times. Figure x shows a 3×3 kernel being used to focus on an image, and the dot product is calculated for each pixel in the area, helping to reduce the dimensions that are outputted. A stride length can also be used in a convolutional layer to change how far the filter moves across the image; by default this is one, meaning the filter will move one pixel at a time. [3]

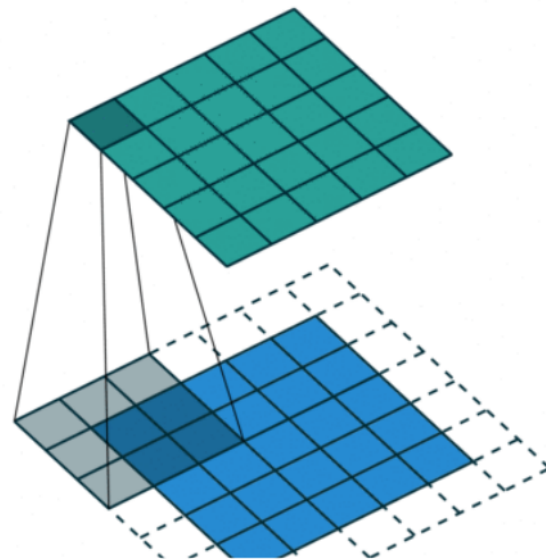


Figure 1 - The Convolutional Layer Kernel moving across and image (Source: https://github.com/vdumoulin/conv_arithmetic)

2.3. Hyper-parameters

There are a number of hyper-parameters that can be used to tune a machine learning model. Some hyper-parameters, such as the kernel/filter size and stride of a convolutional layer, have already been discussed.

2.3.1. Pooling Layer

Pooling layers are used to gradually reduce the dimensionality, thus reducing the number of parameters and the complexity of the model. Pooling layers are generally used after convolutional layers to reduce the dimensionality. Pooling reduces the size of the network by taking a group of pixels in an area (commonly 2x2) and performing an aggregation. Most commonly, this aggregation takes the maximum value of the pixels within the group (known as max-pooling). Max pooling with a 2x2 group can reduce the size to 25%, effectively halving the size of the image. The pooling size is a hyper-parameter in this case; changing it will drastically affect the outcome of a pooling-layer. [4]

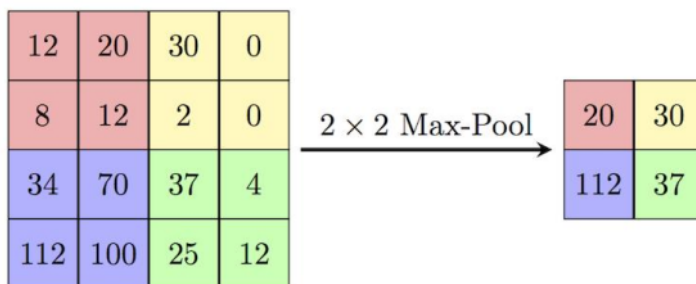


Figure 2 - An example of how max-pooling affects the layer.

2.3.2. Learning Rate

Learning rate is one of the most important hyperparameters to configure; it controls how quickly the neural network learns the problem. The learning rate defines how much the weights are updated during training. The value it is set to is normally a small number between 0.0 and 1.0. With standard gradient descent, the default learning rate is set to 0.01. The optimal learning rate is generally around half of the maximum learning rate. Goodfellow

states, "The learning rate is perhaps the most important hyperparameter. If you have to tune only one hyperparameter, tune the learning rate." [5] , this goes to show just how important it is. [6]

2.3.3. Dropout

Dropout is a widely used regularization technique that is widely used to help fix models that are overfitting. Dropout works by randomly removing units and their connections from the neural network during training. [7] Dropping these units discourages co-adaption between the neurons, resulting in more robust feature extraction. It effectively removed some of the learned features from the network, this helps when the model is overfitting as it removes a lot of the data that it is fitting to the training data specifically, allowing it to generalize better and help with the validation accuracy. [8]

2.3.4. Activation function

The activation function used in each layer can be considered a hyper-parameter. Activation functions are used to help the network learn complex patterns; they introduce nonlinearity between the layers of a neural network. Without activation functions, chaining the layers would result in another linear transformation. Without an activation function, even a deep stack of layers would end up being equivalent to a single layer, which cannot be used to solve complex problems. There are many different types of activation functions, such as sigmoid, tanh, ReLu (Rectified Linear Unit), and softmax. In general, ReLu is used for the hidden layers of a CNN as it is quick to compute. The output layer generally used softmax as it outputs a probability distribution for each class, the sum of values being 1.

3. Experiments

3.1. Examining the Data

The dataset used for the CNN contained greyscale 48x48 pixel images. The data we already partitioned into training and testing sets, therefore only a validation set had to be made. When importing the images into TensorFlow, they were set to a single channel for greyscale, and the color values were normalized. These images were small enough to not require resizing.

3.2. Performance Measure

A performance measure had to be selected for the model, a validation set was made, set to 20% of the training data. Standard Gradient Descent with momentum was used with the model. A separate validation set must be used to prevent over-fitting of the model.



3.3. Testing Model Architecture

The model architecture, e.g., the layers the make up the model, would have to be tried, tested, and edited to get a good result. This would mean training a basic model to start, then adding or removing layers, or tuning hyper-parameters based on their results. In this case, many different models were tried before the final model was decided on.

3.3.1. Starting Architecture

A basic model architecture was used for the first model; it consisted of three convolutional layers, a max-pooling layer, and a dropout layer. Softmax was used for the output layer as it allows the model to generate probabilities for multiple classes, which was required in this case since there were four classes. The order for the layers can be seen in figure 3. These layers were chosen as they provide a good starting point with a nice mix of layers.

```
model = Sequential([
    Conv2D(32, 3, activation='relu', padding='same', input_shape=(48,48,1)),
    Conv2D(32, 3, activation='relu', padding='same'),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),
    Conv2D(32, 3, activation='relu', padding='same'),
    Flatten(),
    # Use softmax here to allow it to generate probabilities for multiple classes
    Dense(4, activation='softmax')
])
```

Figure 3 - The layer setup for a basic CNN

This model, although basic, provided reasonably good accuracy and validation accuracy. After 15 epochs, the accuracy was at 0.63 (63%), and the validation accuracy was at 0.56 (56%). This means the model was slightly overfitting.

3.3.2. Increasing the Learning Rate

To try and improve the accuracy, the training rate was increased from 0.01 to 0.02. The rest of the model stayed the same. The increase in training rate made the model higher in training accuracy but the validation accuracy actually decreased. This can be seen in the Tensor Board chart in figure x.



Figure 4 - A Tensor Board chart showing how learning rate affects the model accuracy

3.3.3. Increasing the Hidden Layers

A new model was created, this time using some more hidden layers, alternating a convolutional layer then a max-pooling layer straight after; the idea for this was to reduce the overfitting. This model still overfitted a little. The end result after 15 epochs was an accuracy of 0.7073 and a validation accuracy of 0.653. This was a fairly large improvement over the basic model, without increasing the complexity of the model too much.

3.3.4. Increasing the number of Convolutional Layers

In an attempt to increase the accuracy of the model even further, some additional

convolutional layers were added. The dropout was also lowered to 0.2 from 0.25. The extra complexity helped a little; the final results came in with an accuracy of 0.7171 and a validation accuracy of 0.6027, meaning it was still overfitting somewhat.

3.3.5. Convolutional Strides and Early Stopping

For the next model, a stride of 2 was added to some of the convolutional layers. The number of epochs was increased to 100 in an attempt to find the optimal number. An early stopping callback was added to stop the model from training once there had been no improvement for three epochs; this would prevent the model from pointlessly training and overfitting. The early stopping callback stopped the training on epoch 54, with an accuracy of 0.9229 and a validation accuracy of 0.6379, meaning it had overfitted somewhat. A chart was made to plot the results, these charts showed the model started overfitting after epoch 12, and the validation accuracy started fluctuating highly in the 60s, with no improvement. The conclusion of this is that the epochs should be kept between 10 and 15.

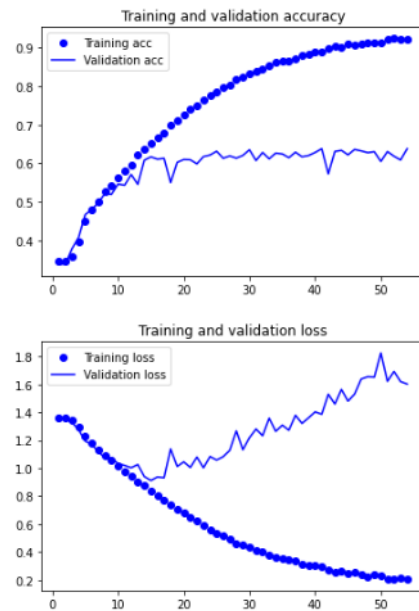


Figure 5 - Training and Validation accuracy plotted for a larger Neural Network

3.3.6. Decreased Dropout

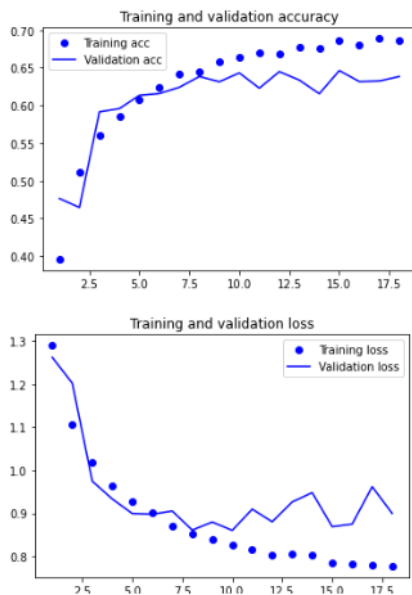


Figure 6 - The Training and Validation accuracy after decreasing the dropout

3.3.7. Final Model Architecture

The final model was very similar to the last but increased the first dropout layer from 0.2 to 0.5, and it also included a dense layer with 128 units. This model was the best; it stopped most of the overfitting and provided a good result. The accuracy score was 0.61, and the validation accuracy was 0.63. This model could be almost considered under fitted, but the testing and validation accuracy was the closest out of all the tests, so it was chosen as the final model. This model was saved so it could be tested using a separate test set.

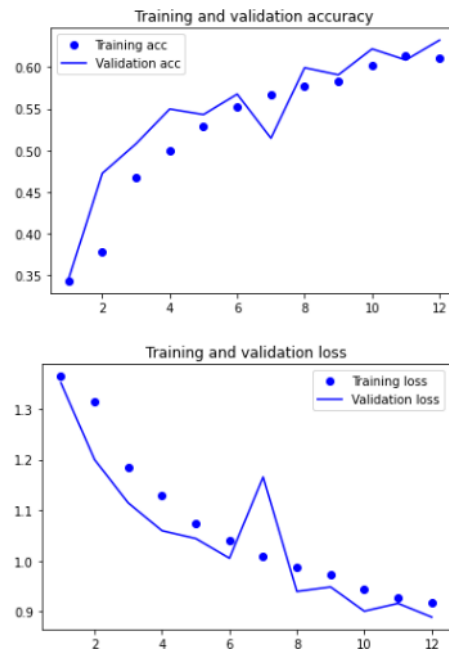


Figure 7 - The Training and Validation accuracy of the finalized neural network architecture.

3.4. Evaluating the Model

The model was saved so that it could be evaluated using the separate test data; this would be a more realistic accuracy reading. The model was tested at a batch size of 16, and the resulting accuracy was 63% (0.6307). This is a very good result since it is almost identical to the validation accuracy when training the model. Although the accuracy of this number is fairly low, it's possible there are some issues with the dataset, such as issues with the images or simply not a large enough dataset for the task.

```
In [36]: test_set.reset()
test_set.batch_size=16

test_loss, test_acc = new_model.evaluate(test_set);

WARNING:tensorflow:sample_weight modes were coerced from
...
to
['...']
326/326 [-----] - 6s 19ms/step - loss: 0.8943 - accuracy: 0.6307
```

Figure 8 - The Final Model accuracy tested using a separate test set

4. Conclusion

This report has outlined the main features of Deep Neural Networks and Convolutional Neural Networks, exploring some of the layers that can be implemented in each and the hyper-parameters that can be tuned. An experimental Convolutional Neural Network has also been created and shown to be fairly successful at detecting facial expressions in images. More work could be done with this model and the data to allow it to detect the expressions at a higher accuracy level.

Bibliography

- [1] S. S. K. Sarvepalli, "Deep Learning in Neural Networks: The science behind an Artificial Brain," Liverpool Hope University, Liverpool , 2017.
- [2] K. O'Shea and R. Nash, "An Introduction to Convolutional Neural Networks," Lancaster University, Lancaster, 2015.
- [3] G. W. Lindsay, PhD, "Convolutional Neural Networks as a Model of the Visual," Gatsby Computational Unit, London.
- [4] S. Albawi, T. Abed Mohammed and S. ALZAWI, "Understanding of a Convolutional Neural Network," 2017.
- [5] I. Goodfellow, Deep Learning (Adaptive Computation and Machine Learning series), Massachusetts: MIT Press, 2017.
- [6] Y. Bengio, "Practical Recommendations for Gradient-Based Training of Deep," Arxiv, 2012.
- [7] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from," *Journal of Machine Learning Research*, no. 15, pp. 1929 - 1958, 2014.
- [8] S. G. J. Z. M. W. W. C. G. O. B. Cai, Effective and Efficient Dropout for Deep Convolutional Neural Networks, 2019.

Appendices

A. Model Training Code

```
import tensorflow as tf
import numpy as np
from tensorflow import keras

import pathlib

print(tf.__version__)
print(keras.__version__)

img_height = 48
img_width = 48
batch_size = 16

data_dir = pathlib.Path('./dataset/train')

image_count = len(list(data_dir.glob('*/*.jpg')))

print(image_count)

# Load the dataset/filenames into a list
```

```
list_ds = tf.data.Dataset.list_files(str(data_dir/'*/'*'), shuffle=False)

# Shuffle the dataset
list_ds = list_ds.shuffle(image_count, reshuffle_each_iteration=False)

# Check the data has been loaded by looking at the first 5 entries in the list

for f in list_ds.take(5):
    print(f.numpy())

# Get the class names from the folder names, store in an array

class_names = np.array(sorted([item.name for item in data_dir.glob('*')]))
print(class_names)

# Create the validation set size, 20% of the total image count
val_size = int(image_count * 0.2)

# Create a train dataset
train_ds = list_ds.skip(val_size)

# Create a validation dataset, set to the size we created earlier
val_ds = list_ds.take(val_size)

# Print the size of the training dataset
```

```
print(tf.data.experimental.cardinality(train_ds).numpy())

# Print the size of the validation dataset
print(tf.data.experimental.cardinality(val_ds).numpy())

import os

# Get the label of a dataset entry
def get_label(file_path):
    # convert the path to a list of path components
    parts = tf.strings.split(file_path, os.path.sep)
    # The second to last is the class-directory

    #convert the bool result to in
    one_hot = tf.dtypes.cast(parts[-2] == class_names, dtype = tf.int16)
    # Integer encode the label
    return tf.argmax(one_hot)

# Decode the image
def decode_img(img):
    # convert the compressed string to a 3D uint8 tensor
    img = tf.image.decode_jpeg(img, channels=1)

    # Scale the RGB values and convert to float32 (Scaled 0-1 instead of 0-255)
    img = tf.image.convert_image_dtype(img, tf.float32)
```

```
# resize the image to the desired size
return tf.image.resize(img, [img_height, img_width])

# Get the image, path values for a dataset entry
def process_path(file_path):
    label = get_label(file_path)
    # load the raw data from the file as a string
    img = tf.io.read_file(file_path)
    img = decode_img(img)
    return img, label

# Create a new dataset, mapping the images and labels
# num_parallel_calls=AUTOTUNE allows this to be done in parallel to save time

AUTOTUNE = tf.data.experimental.AUTOTUNE

train_ds = train_ds.map(process_path, num_parallel_calls=AUTOTUNE)
val_ds = val_ds.map(process_path, num_parallel_calls=AUTOTUNE)

# Check the images shape and one hot encoding has worked
for image, label in train_ds.take(2):
    print("Image shape:", image.numpy().shape)
    print("Label:", label.numpy())
```

```
# Configure the dataset for performance
# Shuffle the dataset
# Batch the dataset
# Prefetch buffer

# More info at https://www.tensorflow.org/guide/data\_performance
def configure_for_performance(ds):
    ds = ds.cache()
    ds = ds.shuffle(buffer_size=1000)
    ds = ds.batch(batch_size)
    ds = ds.prefetch(buffer_size=AUTOTUNE)
    return ds

train_ds = configure_for_performance(train_ds)
val_ds = configure_for_performance(val_ds)

image_batch, label_batch = next(iter(train_ds))

import matplotlib.pyplot as plt

first_img = image_batch[0]
print(np.min(image_batch), np.max(image_batch))

plt.figure(figsize=(9,9))
for i in range(9):
```

```
plt.subplot(3,3,i+1)
plt.xticks([])
plt.yticks([])
plt.grid(False)
plt.imshow(image_batch[i], cmap='gray')
plt.xlabel(class_names[label_batch[i]])
plt.show()

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

model = Sequential([
    Conv2D(32, 3, activation='relu', padding='same', input_shape=(48,48,1)),
    Conv2D(32, 3, activation='relu', padding='same'),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),
    Conv2D(32, 3, activation='relu', padding='same'),
    Flatten(),
    # Use softmax here to allow it to generate probabilities for multiple classes
    Dense(4, activation='softmax')
])

model.summary()
```

```
model.compile(loss='sparse_categorical_crossentropy',optimizer=keras.optimizers.SGD(lr=0.01),  
metrics=["accuracy"])
```

```
# Create a log directory
```

```
root_logdir = os.path.join(os.getcwd(),"logs_batch16_2")
```

```
def get_run_logdir():
```

```
    import time
```

```
    run_id=time.strftime("run_%Y_%m_%d_%H_%M_%S")
```

```
    return os.path.join(root_logdir,run_id)
```

```
run_logdir = get_run_logdir() + "_model_1"
```

```
print(run_logdir)
```

```
#Create a Tensorboard callback function that uses the log directory
```

```
tensorboard_cb=keras.callbacks.TensorBoard(run_logdir)
```

```
#Train the Model - Specify the callback to place logs in the log folder we created
```

```
history=model.fit(train_ds,epochs=15,
```

```
    validation_data=val_ds,
```

```
    callbacks=[tensorboard_cb])
```

```
from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

model = Sequential([
    Conv2D(32, 3, activation='relu', padding='same', input_shape=(48, 48, 1)),
    Conv2D(32, 3, activation='relu', padding='same'),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),
    Conv2D(32, 3, activation='relu', padding='same'),
    Flatten(),
    # Use softmax here to allow it to generate probabilities for multiple classes
    Dense(4, activation='softmax')
])

model.summary()

model.compile(loss='sparse_categorical_crossentropy', optimizer=keras.optimizers.SGD(lr=0.02),
metrics=["accuracy"])

run_logdir = get_run_logdir()+"_model_1_inc_lr"

print(run_logdir)

# Create a Tensorboard callback function that uses the log directory
tensorboard_cb=keras.callbacks.TensorBoard(run_logdir)
```

#Train the Model - Specify the callback to place logs in the log folder we created

```
history=model.fit(train_ds,epochs=15,  
                  validation_data=val_ds,  
                  callbacks=[tensorboard_cb])
```

```
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,Dropout
```

```
model = Sequential([  
    Conv2D(32, 3, activation='relu', padding='same',input_shape=(48,48,1)),  
  
    Conv2D(32, 3, activation='relu', padding='same'),  
    MaxPooling2D(pool_size=(2, 2)),  
  
    Conv2D(64, 3, activation='relu', padding='same'),  
    MaxPooling2D(pool_size=(2, 2)),  
  
    Conv2D(128, 3, activation='relu', padding='same'),  
    MaxPooling2D(pool_size=(2, 2)),  
    Dropout(0.25),  
  
    Flatten(),
```

```
# Use softmax here to allow it to generate probabilities for multiple classes
Dense(4, activation='softmax')
])
model.summary()
model.compile(loss='sparse_categorical_crossentropy',optimizer=keras.optimizers.SGD(lr=0.02),
metrics=["accuracy"])
run_logdir = get_run_logdir() + "_model_2"

print(run_logdir)
#Create a Tensorboard callback function that uses the log directory
tensorboard_cb=keras.callbacks.TensorBoard(run_logdir)

#Train the Model - Specify the callback to place logs in the log folder we created

history=model.fit(train_ds,epochs=15,
                  validation_data=val_ds,
                  callbacks=[tensorboard_cb])

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,Dropout,
BatchNormalization

model = Sequential([
    Conv2D(32, 3, activation='relu', padding='same',input_shape=(48,48,1)),
```

```
Conv2D(32, 3, activation='relu', padding='same'),
Conv2D(32, 3, activation='relu', padding='same'),
MaxPooling2D(pool_size=(2, 2)),

Conv2D(64, 3, activation='relu', padding='same'),
Conv2D(64, 3, activation='relu', padding='same'),
MaxPooling2D(pool_size=(2, 2)),

Dropout(0.2),

Conv2D(128, 3, activation='relu', padding='same'),
Conv2D(128, 3, activation='relu', padding='same'),
MaxPooling2D(pool_size=(2, 2)),

Dropout(0.2),

Flatten(),

# Use softmax here to allow it to generate probabilities for multiple classes
Dense(4, activation='softmax')
])
```

```
model.compile(loss='sparse_categorical_crossentropy',optimizer=keras.optimizers.SGD(lr=0.02),
metrics=["accuracy"])

run_logdir = get_run_logdir()+"_model_3"

print(run_logdir)

#Create a Tensorboard callback function that uses the log directory

tensorboard_cb=keras.callbacks.TensorBoard(run_logdir)


#Train the Model - Specify the callback to place logs in the log folder we created


history=model.fit(train_ds,epochs=15,
                  validation_data=val_ds,
                  callbacks=[tensorboard_cb])

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,Dropout,
BatchNormalization


model = Sequential([

    Conv2D(32, 3, activation='relu', padding='same',input_shape=(48,48,1)),

    Conv2D(32, 3, activation='relu', padding='same'),

    Conv2D(32, kernel_size=(3, 3), strides=2, activation='relu', padding='same'),

    MaxPooling2D(pool_size=(2, 2)),

    Conv2D(64, 3, activation='relu', padding='same'),
```

```
Conv2D(64, 3, activation='relu', padding='same'),
MaxPooling2D(pool_size=(2, 2)),
Dropout(0.2),
Conv2D(128, 3, activation='relu', padding='same'),
Conv2D(128, kernel_size=(3, 3), strides=2, activation='relu', padding='same'),
MaxPooling2D(pool_size=(2, 2)),
Dropout(0.2),
Flatten(),
# Use softmax here to allow it to generate probabilities for multiple classes
Dense(4, activation='softmax')
])
model.summary()
model.compile(loss='sparse_categorical_crossentropy', optimizer=keras.optimizers.SGD(lr=0.02),
metrics=["accuracy"])
run_logdir = get_run_logdir()+"_model_final"

print(run_logdir)
#Create a Tensorboard callback function that uses the log directory
tensorboard_cb=keras.callbacks.TensorBoard(run_logdir)
#Create an EarlyStopping callback, will stop once there has been no improvement for 3
epochs
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)
#Train the Model - Specify the callback to place logs in the log folder we created

history=model.fit(train_ds, epochs=100,
validation_data=val_ds,
```

```
callbacks=[tensorboard_cb, callback])  
print(model.history.history.keys())  
  
acc = model.history.history['accuracy']  
val_acc = model.history.history['val_accuracy']  
loss = model.history.history['loss']  
val_loss = model.history.history['val_loss']  
  
epochs = range(1, len(acc) + 1)  
  
plt.plot(epochs, acc, 'bo', label='Training acc')  
plt.plot(epochs, val_acc, 'b', label='Validation acc')  
plt.title('Training and validation accuracy')  
plt.legend()  
  
plt.figure()  
  
plt.plot(epochs, loss, 'bo', label='Training loss')  
plt.plot(epochs, val_loss, 'b', label='Validation loss')  
plt.title('Training and validation loss')  
plt.legend()  
  
plt.show()
```

```
from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout,
BatchNormalization


model = Sequential([

    Conv2D(32, 3, activation='relu', padding='same', input_shape=(48,48,1)),

    Conv2D(32, 3, activation='relu', padding='same'),
    Conv2D(32, kernel_size=(3, 3), strides=2, activation='relu', padding='same'),
    MaxPooling2D(pool_size=(2, 2)),

    Conv2D(64, 3, activation='relu', padding='same'),
    Conv2D(64, 3, activation='relu', padding='same'),
    MaxPooling2D(pool_size=(2, 2)),

    Dropout(0.3),

    Conv2D(128, 3, activation='relu', padding='same'),
    Conv2D(128, kernel_size=(3, 3), strides=2, activation='relu', padding='same'),
    MaxPooling2D(pool_size=(2, 2)),

    Dropout(0.2),
```

```
Flatten(),

# Use softmax here to allow it to generate probabilities for multiple classes
Dense(4, activation='softmax')
])
model.summary()
model.compile(loss='sparse_categorical_crossentropy', optimizer=keras.optimizers.SGD(lr=0.01),
metrics=["accuracy"])
run_logdir = get_run_logdir()+"_model_final2"
print(run_logdir)

#Create a Tensorboard callback function that uses the log directory
tensorboard_cb=keras.callbacks.TensorBoard(run_logdir)

#Create an EarlyStopping callback, will stop once there has been no improvement for 3
epochs
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)

#Train the Model - Specify the callback to place logs in the log folder we created
history=model.fit(train_ds, epochs=100,
                  validation_data=val_ds,
                  callbacks=[tensorboard_cb, callback])
print(model.history.history.keys())

acc = model.history.history['accuracy']
```

```
val_acc = model.history.history['val_accuracy']
loss = model.history.history['loss']
val_loss = model.history.history['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout,
BatchNormalization
```

```
model = Sequential([
    Conv2D(32, 3, activation='relu', padding='same', input_shape=(48,48,1)),

    Conv2D(32, 3, activation='relu', padding='same'),
    Conv2D(32, kernel_size=(3, 3), strides=2, activation='relu', padding='same'),
    MaxPooling2D(pool_size=(2, 2)),

    Dropout(0.25),

    Conv2D(64, 3, activation='relu', padding='same'),
    Conv2D(64, 3, activation='relu', padding='same'),
    MaxPooling2D(pool_size=(2, 2)),

    Dropout(0.25),

    Conv2D(128, 3, activation='relu', padding='same'),
    Conv2D(128, kernel_size=(3, 3), strides=2, activation='relu', padding='same'),
    MaxPooling2D(pool_size=(2, 2)),

    Flatten(),

    # Use softmax here to allow it to generate probabilities for multiple classes
    Dense(4, activation='softmax')
```

```
    ])  
    model.summary()  
    model.compile(loss='sparse_categorical_crossentropy', metrics=['accuracy'])  
    run_logdir = get_run_logdir()+"_model_final3"  
  
    print(run_logdir)  
    #Create a Tensorboard callback function that uses the log directory  
    tensorboard_cb=keras.callbacks.TensorBoard(run_logdir)  
  
    #Create an EarlyStopping callback, will stop once there has been no improvement for 3  
    epochs  
    callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)  
  
    #Train the Model - Specify the callback to place logs in the log folder we created  
  
    history=model.fit(train_ds, epochs=18,  
                      validation_data=val_ds,  
                      callbacks=[tensorboard_cb, callback])  
    acc = model.history.history['accuracy']  
    val_acc = model.history.history['val_accuracy']  
    loss = model.history.history['loss']  
    val_loss = model.history.history['val_loss']  
  
    epochs = range(1, len(acc) + 1)
```

```
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout,
BatchNormalization

finalModel = Sequential([
    Conv2D(32, 3, activation='relu', padding='same', input_shape=(48,48,1)),

    Conv2D(32, 3, activation='relu', padding='same'),
    Conv2D(32, kernel_size=(3, 3), strides=2, activation='relu', padding='same'),
    MaxPooling2D(pool_size=(2, 2)),
```

```
Dropout(0.5),

Conv2D(64, 3, activation='relu', padding='same'),
Conv2D(64, 3, activation='relu', padding='same'),
MaxPooling2D(pool_size=(2, 2)),

Dropout(0.25),

Conv2D(128, 3, activation='relu', padding='same'),
Conv2D(128, kernel_size=(3, 3), strides=2, activation='relu', padding='same'),
MaxPooling2D(pool_size=(2, 2)),

Flatten(),

Dense(128),

# Use softmax here to allow it to generate probabilities for multiple classes
Dense(4, activation='softmax')
])
finalModel.summary()
finalModel.compile(loss='sparse_categorical_crossentropy', metrics=["accuracy"])
run_logdir = get_run_logdir()+"_model_finalF"
```

```
print(run_logdir)

#Create a Tensorboard callback function that uses the log directory
tensorboard_cb=keras.callbacks.TensorBoard(run_logdir)

#Create an EarlyStopping callback, will stop once there has been no improvement for 3
epochs

callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)

#Train the Model - Specify the callback to place logs in the log folder we created

history=finalModel.fit(train_ds,epochs=12,
                        validation_data=val_ds,
                        callbacks=[tensorboard_cb, callback])

acc = finalModel.history.history['accuracy']
val_acc = finalModel.history.history['val_accuracy']
loss = finalModel.history.history['loss']
val_loss = finalModel.history.history['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
```

```
plt.legend()
```

```
plt.figure()
```

```
plt.plot(epochs, loss, 'bo', label='Training loss')
```

```
plt.plot(epochs, val_loss, 'b', label='Validation loss')
```

```
plt.title('Training and validation loss')
```

```
plt.legend()
```

```
plt.show()
```

```
model.save("final_model.h5")
```

Machine Learning 2

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

68/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5



Comment 1

ref is what? - double check doc before submitting

PAGE 6

PAGE 7

PAGE 8



Comment 2

Explain the performance measure here - that is what is being measured. Show the calculation.

PAGE 9



Comment 3

can you show any refs that support this approach?



Comment 4

Good. Well done. What about the test set performance?

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32

PAGE 33

PAGE 34

PAGE 35

PAGE 36

WILLIAM HENRY BAKER

67.50

RESEARCH-INFORMED LITERATURE



A fair review of concepts with refs used appropriately. A look at dataset pre-processing especially identifying biases in the datasets would have been useful later on.

6

KNOWLEDGE AND UNDERSTANDING



Explain the performance measure- that is, what is being measured. Show the calculation. Explain the idea of the validation set in a little more detail.

11

ANALYSIS



When you modify your design try to show any refs that support your design choices. You could/should have also explored the accuracy for each class. The dataset had some biases in it - too. Some emotions were over or under represented in the training and test set. You need to account for these. No detailed multi class evaluation here. Otherwise this would be an excellent submission.

20

PRACTICAL APPLICATION AND DEP.

 Code developed to create experiments. NN designs and implementations present. Training and validation sets incorporated into the process. Test sets used for evaluation. Good.

27

SKILLS FOR PROFESSIONAL PRAC.

 Report structured and refs used appropriately. Diagrams helpful.

3.5