TENSORFLOW

CHEAT SHEET

for Deep Learning Model Building

01 FEEDFORWARD NEURAL NETWORK

```
model = models.Sequential()
model.add(layers.Flatten(input_shape=(input_size,)))  # Adjust input_size based on your data

# Add hidden layers
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dropout(0.2))  # Optional: Add dropout for regularization

# Add output layer
model.add(layers.Dense(output_size, activation='softmax'))  # Adjust output_size based on your problem

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',  # Use 'categorical_crossentropy' for one-hot encoded labels
              metrics=['accuracy'])
```

02 CONVOLUTIONAL NEURAL NETWORK

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
input_shape=(img_height, img_width, channels)))
model.add(layers.MaxPooling2D((2, 2)))

# Add more convolutional and pooling layers as
needed

model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(output_size,
activation='softmax'))

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

03 RECURRENT NEURAL NETWORK

```
model = models.Sequential()
model.add(layers.SimpleRNN(128,
activation='relu', input_shape=(timesteps,
features)))

# Add more recurrent layers or use LSTM/GRU
layers

model.add(layers.Dense(output_size,
activation='softmax'))

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

04 LONG SHORT-TERM MEMORY

```
model = models.Sequential()
model.add(layers.LSTM(128, activation='relu',
input_shape=(timesteps, features)))

# Add more LSTM layers if needed

model.add(layers.Dense(output_size,
activation='softmax'))

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

05 GATED RECURRENT UNIT

```
model = models.Sequential()
model.add(layers.GRU(128, activation='relu',
input_shape=(timesteps, features)))

# Add more GRU layers if needed

model.add(layers.Dense(output_size,
activation='softmax'))

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

06 TRANSFER LEARNING(E.G., VGG16)

```
from tensorflow.keras.applications import VGG16

# Load pre-trained VGG16 model without the top layer
base_model = VGG16(weights='imagenet', include_top=False,
input_shape=(img_height, img_width, channels))

# Freeze convolutional layers
for layer in base_model.layers:
    layer.trainable = False

model = models.Sequential()
model.add(base_model)

# Add custom classification layers
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(output_size, activation='softmax'))

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

07 BATCH NORMALIZATION

```
model.add(layers.BatchNormalization())
```

08 DATA AUGMENTATION

```
from tensorflow.keras.preprocessing.image import
ImageDataGenerator

datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    shear_range=0.2
)

datagen.fit(X_train)  # X_train is your training data

model.fit(datagen.flow(X_train, y_train, batch_size=batch_size),
epochs=epochs)
```

09 EARLY STOPPING

```
from tensorflow.keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(monitor='val_loss', patience=3,
restore_best_weights=True)

model.fit(X_train, y_train, epochs=epochs, validation_data=
(X_val, y_val), callbacks=[early_stopping])
```

10 LEARNING RATE SCHEDULER

```
from tensorflow.keras.callbacks import LearningRateScheduler

def scheduler(epoch, lr):
    if epoch % 10 == 0 and epoch != 0:
        return lr * 0.9
    else:
        return lr

lr_scheduler = LearningRateScheduler(scheduler)

model.fit(X_train, y_train, epochs=epochs, validation_data=
(X_val, y_val), callbacks=[lr_scheduler])
```

11 EARLY STOPPING

```
from sklearn.model_selection import GridSearchCV
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier

# Define your model creation function
def create_model(optimizer='adam', hidden_units=128, dropout_rate=0.2):
    model = models.Sequential()
    model.add(layers.Flatten(input_shape=(input_size,)))

    # Add hidden layers
    model.add(layers.Dense(hidden_units, activation='relu'))
    model.add(layers.Dropout(dropout_rate))

    # Add output layer
    model.add(layers.Dense(output_size, activation='softmax'))

    model.compile(optimizer=optimizer,
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model

# Create a KerasClassifier with your model creation function
model = KerasClassifier(build_fn=create_model, epochs=10, batch_size=32, verbose=0)

# Define the hyperparameters to search
param_grid = {
    'optimizer': ['adam', 'sgd', 'rmsprop'],
    'hidden_units': [64, 128, 256],
    'dropout_rate': [0.2, 0.5, 0.8]
}

# Use GridSearchCV for hyperparameter search
grid = GridSearchCV(estimator=model, param_grid=param_grid, cv=3)
grid_result = grid.fit(X_train, y_train)

# Print the best parameters and corresponding accuracy
print("Best Parameters: ", grid_result.best_params_)
print("Best Accuracy: ", grid_result.best_score_)
```